Universidad
Carlos III de Madrid

AUTOMATIZACIÓN DE PROCESO DE ATORNILLADO MECÁNICO A TRAVÉS DE ELEMENTOS DE CONTROL ELECTRÓNICO

Realizado por: D. José Félix Marín Blanca
Tutor de proyecto: Dra. Carolina Álvarez Caldas
Ingeniería Técnica Industrial esp. Mecánica

17 de diciembre de 2010

INDICE PROYECTO

INDICE DE TABLAS

INDICE DE FIGURAS

Dedicatoria:

A mi familia y futura familia (si Dios quiere)…

1. Introducción

1.1. Contexto del proyecto:

Un **proceso de fabricación**, también denominado **proceso industrial**, *manufactura* o *producción*, es el conjunto de operaciones necesarias para modificar las características de las materias primas. Dichas características pueden ser de naturaleza muy variada tales como la forma, la densidad, la resistencia, el tamaño o la estética. Se realizan en el ámbito de la industria. Para realizar dichas operaciones con objetivos de diversa consideración (plazos, calidad, etc…) se necesita de maquinaria adecuada. Uno de estos procesos es el atornillado

Este proyecto contempla la posibilidad de automatizar un proceso industrial mecánico a través de elementos estándar del mercado, tanto para el control propio del proceso, así como la flexibilización del mismo y la garantía de los más exigentes requisitos de calidad en el proceso.

1.2. Motivación del proyecto

Se elige este tema como proyecto debido a la gran necesidad de la industria de garantizar la precisión de fabricación de algunos de los procesos mecánicos más elementales y para poder demostrar alguna de las relaciones de dependencia entre elementos mecánicos y electrónicos.

1.3. Estructura del proyecto

En este proyecto se van a plantear los objetivos generales del proyecto, así como la arquitectura propuesta para realizar dichos objetivos. Así mismo, se describirán las partes que componen cada una de las partes del sistema, las cuales consisten en el PLC (autómata programable), sistemas de comunicación (Modbus y Canopen), sistema de supervisión HMI (XBTGT), así como los software de programación Twidosuite y Vijeo Designer. También se describirá el funcionamiento del atornillador automático y las posibles configuraciones del mismo. Se hará una explicación del sistema y se incluirá los documentos que contienen la programación y detalles del sistema.

2. Objetivos

En este proyecto se va a tratar de demostrar que es posible la integración de elementos para mejorar el proceso de atornillado automático y cumplir con las necesidades que exige el mercado actualmente. Se hará de la siguiente forma:

- Se trata de integrar y supervisar el proceso de atornillado, configurando una nueva máquina industrial, a través de controles estándar de mercado. Para ello, se integrará un PLC para el control global de la máquina, un encóder de comunicación industrial para transmitir la posición del atornillador y pantalla HMI (Human Machine Interface) para el control y supervisión del proceso, y un atornillador automático para desarrollar la parte del atornillado mecánico del proceso.

Para conseguir dicho objetivo, se pretende:

- Definir y desarrollar el funcionamiento de un PLC y su entorno de programación.
- Definición y desarrollo de los buses de comunicación utilizados, en este caso, Modbus/JBus para el terminal gráfico y CanOpen para los encoder de captación de movimiento.
- Definir y desarrollar el concepto de HMI con un terminal táctil y su entorno de programación.
- Definir y desarrollar los encoder de captación de movimiento así como su funcionamiento
- Definir y desarrollar el funcionamiento de un sistema de atornillado automático.

Se realizará un ejemplo práctico donde se desarrolla un sistema integral, el cual se compondrá de los elementos anteriormente definidos y permitirá realizar todas las especificaciones del proyecto.

En el presente proyecto también se hablará de:

- El desarrollo y programación necesaria del control PLC para conseguir los objetivos definidos.
- El desarrollo y programación del terminal HMI, definiendo la base interfaz con el usuario.
- Implementación de los distintos tipos de comunicación industrial utilizados en la aplicación, tanto para el sistema PLC-encoder como para el sistema PLC-HMI.

3. Descripción de los componentes desarrollados en el sistema

3.1. Arquitectura propuesta

En el proyecto se pretende realizar una red de comunicación para desarrollar una completa integración de los equipos en un sistema. Para ello se utilizarán una serie de equipos que completan una red genérica. A continuación se hace una breve descripción de cómo será el sistema final:

- Un PLC gobierna el control del atornillador automático a través de Entradas/Salidas que compartirá con el atornillador. Dos encoder con comunicación CANOPEN gobernados por el PLC darán la posición del atornillador en los ejes X e Y. Un terminal gráfico de 5,7" táctil comunicado a través de MODBUS con el PLC, hará las funciones de supervisión y control por parte del operario, controlando las acciones del PLC y determinando las opciones del proceso. La arquitectura propuesta es la siguiente:

El objetivo final de esta aplicación es controlar el proceso de atornillado, tanto la posición de la operación como el orden atornillado. Es una aplicación parametrizada que tiene en cuenta variables como el par de apriete o la posición de los tornillos a atornillar. El operario podrá decidir cuál va a ser el orden de la operación, el par de apriete y desde el control HMI puede supervisar que toda la operación se está desarrollando de forma correcta.

Cualquiera de las operaciones descritas poseerá el mayor grado de flexibilidad posible, ya que se desarrollará con elementos estándar de mercado, fácilmente ampliables en opciones y/o características. En los puntos siguientes se desarrollarán cada unos de los componentes necesarios para la aplicación.

3.2. Sistemas Automatizados. Introducción al PLC.

Se entiende por Autómata Programable, o PLC (Controlador Lógico Programable), toda máquina electrónica, diseñada para controlar en tiempo real y en medio industrial procesos secuenciales. Su manejo y programación puede ser realizada por personal eléctrico o electrónico sin conocimientos informáticos. Realiza funciones lógicas: series, paralelos, temporizaciones, contajes y otras más potentes como cálculos, regulaciones, etc.

Dentro de la definición de autómata programable entra el concepto de «caja negra» en la que existen, por una parte, unos terminales de entrada (o captadores) a los que se conectan cualquier captador de señal digital (valor todo/nada) como por ejemplo pulsadores, finales de carrera, fotocélulas, detectores...; y por otra, unos terminales de salida (o actuadores) a los que se conectarán los elementos físicos sobre los que actuar, como por ejemplo bobinas de contactores, electroválvulas, lámparas., etc… de forma que la actuación de estos últimos está en función de las señales de entrada que estén activadas en cada momento, según el programa almacenado.

La función básica de los autómatas programables es la de flexibilizar los procesos industriales, reduciendo el trabajo del usuario a realizar el *programa*, es decir, la relación entre las señales de entrada que se tienen que cumplir para activar cada salida, teniendo la ventaja que los elementos tradicionales eléctricos (como relés auxiliares, de enclavamiento, temporizadores, contadores...) no son elementos físicos, sino que se encuentran en formato software interno en el PLC.

El autómata es la primera máquina con lenguaje, es decir, un calculador lógico cuyo juego de instrucciones se orienta hacia los sistemas de evolución secuencial.
Hay que apreciar que, cada vez más, la universalidad de los ordenadores tiende a desaparecer. El futuro industrial parece abrirse hacia esta nueva clase de dispositivos: máquina para proceso de señales, para la gestión de bases de datos...
La creciente difusión de aplicaciones de la electrónica, la fantástica disminución del precio de los componentes, el nacimiento y el desarrollo de los microprocesadores y, sobretodo, la miniaturización de los circuitos de memoria permiten presagiar una introducción de los autómatas programables, cuyo precio es atractivo incluso para equipos de prestaciones modestas, en una inmensa gama de nuevos campos de aplicación. Otra de las ventajas de los autómatas programables frente a los ordenadores es su robustez.

El autómata programable satisface las exigencias tanto de procesos continuos como discontinuos. Regula presiones, temperaturas, niveles y caudales así como todas las funciones asociadas de temporización, cadencia, conteo y lógica. También incluye posibilidades de comunicación adicional, con lo que el autómata se transforma en un poderoso satélite dentro de una red de control distribuida.
El autómata programable es un aparato electrónico programable por un usuario programador y destinado a gobernar, dentro de un entorno industrial, maquinas o procesos lógicos secuenciales.
El autómata que se ha escogido para este proyecto es de la marca SCHNEIDER ELECTRIC, concretamente de la gama Twido, que se describirá a continuación.

3.2.1. PLC Twido

3.2.1.1. Presentación de gama

La gama de controladores programables compactos Twido ofrece una solución "todo en uno" con unas dimensiones reducidas: 80 a 157 X 90 X 70 mm. Existen diez controladores compactos, diferenciados por la capacidad de tratamiento y el número de entradas a 24 V, de salida de relé y transistor (10, 16, 24 y 40 entradas/salidas).
Estas bases compactas utilizan:

- una alimentación de corriente alterna comprendida entre a 100 y 240 V (que garantiza la alimentación c 24 V de los captadores)
- una alimentación de corriente continua comprendida entre c 19,2 y 30 V(prever una alimentación auxiliar externa tipo Phaseo para la alimentación de los captadores).

Este tipo de bases compactas presenta las siguientes ventajas:
Una cantidad significativa de entradas/salidas (hasta 40 entradas/salidas) agrupadas en muy poco espacio, lo que permite reducir el tamaño de las consolas o de los cofres en las aplicaciones donde el espacio ocupado resulta primordial. Para los modelos de 24 y 40 entradas/salidas, la posibilidad de ampliar y añadir módulos opcionales ofrece al usuario el mismo grado de flexibilidad que las plataformas de automatismos más importantes:
con las bases compactas de 24 entradas/salidas TWD LCxA 24DRF, hasta 4 módulos de ampliación de entradas/salidas TON, analógicas y/o de comunicación con las bases compactas de 40 entradas/salidas TWD LCxx 40DRF, hasta 7 módulos de ampliación de entradas/salidas TON, analógicas y/o de comunicación.

Módulos opcionales, como visualizador numérico, cartucho de ampliación de memoria, cartucho de reloj calendario y puerto de comunicación RS 485 o RS 232C suplementario.

La solución de controlador compacto aporta también una gran flexibilidad de cableado. Para las ampliaciones de entradas/salidas “Todo o Nada” (con las bases TWD LCxA 24DRF y TWD LCxx 40DRF) se ofrecen varias posibilidades de conexión, como borneros con tornillos desenchufables, conectores de tipo resorte que permiten realizar un cableado sencillo, rápido y seguro.
El sistema de pre-cableado Advantys Telefast ABE 7 permite conectar los módulos con conectores de tipo HE 10 a los cables pre-equipados con hilos libres en uno de los extremos, que al sistema de cableado Advantys Telefast ABE 7 para controlador Twido (conjunto de cables de conexión y bases ABE 7).
El visualizador y la memoria que opcionalmente pueden conectarse a la base facilitan los procesos de ajuste, transferencia y grabación de las aplicaciones el visualizador numérico puede utilizarse como herramienta de visualización y de ajuste local.
La tecnología EEPROM de los cartuchos de memoria permite grabar y transferir programas en cualquier controlador compacto o modular Twido. Con el software TwidoSuite, el cual se presentará más adelante, se puede programar a partir de instrucciones en lenguaje lista de instrucciones o de elementos gráficos en lenguaje de contactos.

3.2.1.2. Introducción al PLC Twido

Para la realización de esta aplicación se ha decidido utilizar autómatas de SCHNEIDER ELECTRIC. Exactamente los equipos Twido TWDLCAA24DRF. Se han elegido estos equipos por la potencia de cómputo de órdenes, por la integración del puerto Modbus RTU en la propia CPU y por la versatilidad de estos sistemas a la hora de configuraciones en red, como CANOpen y por el número de entradas y salidas integradas, entre otras características.

3.2.1.3. Configuración

Debido a la versatilidad entre equipos compactos y modulares de estos equipos se debe llevar un orden a la hora de hacer la configuración. El primer paso es realizar la configuración hardware del proyecto.
Para esta configuración se deben conocer una serie de puntos previos

3.2.1.4. Estructura compacta/modular

El Twido tiene configuración compacta o modular. Es decir, que es posible adquirir CPU´s con entradas y salidas de acceso en la propia CPU, o componerlo de forma personalizada utilizando los distintos módulos del Twido.

Consta de los siguientes módulos:

- CPU con diferentes prestaciones
- Módulos de señales para entradas/salidas digitales y analógicas
- Módulos de función para funciones tecnológicas
- Tarjetas para tareas de comunicación

- Fuentes de alimentación Phaseo de carga para conectar el Twido a una tensión de alimentación de 120/230 V c.a., en caso de escoger una CPU a 24V.
- Todos los módulos del Twido están protegidos por una caja con grado de protección IP 20, es decir, disponen de envolvente y pueden funcionar sin ventilador.

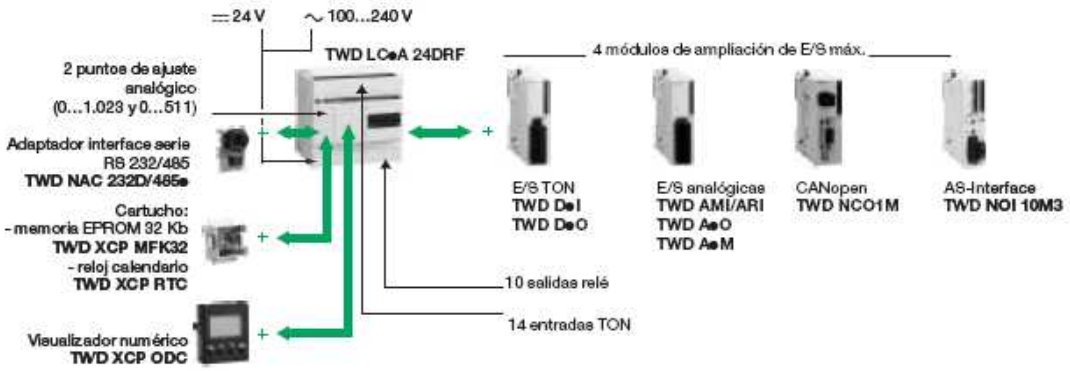
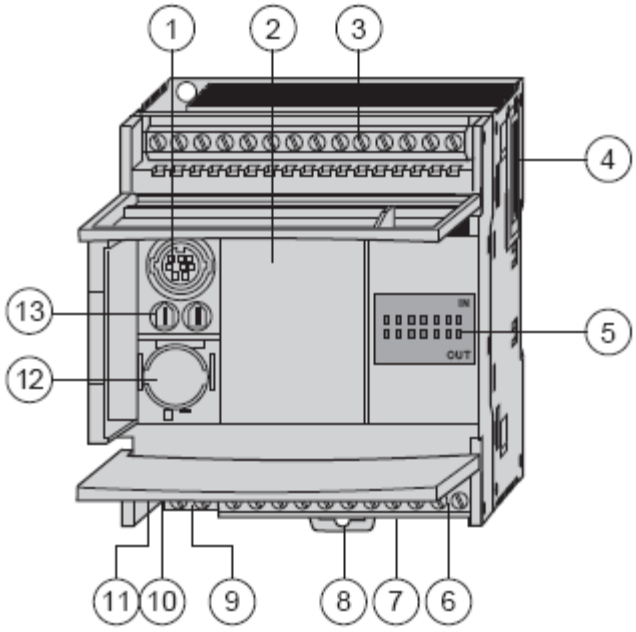


Figura 1. Opciones de componentes de Twido Compacto TWDLCxA24DRF

3.2.1.5. Descripción Hardware

Todos los PLC´s Twido comparten las siguientes caracteristicas:



① - Puerto serie 1
② - Tapa móvil (Pantalla digital)
③ - Bornera de tornillos de alimentación y entradas del detector
④ - Conector de extensión E/S
⑤ - Bloque de visualización del estado del controlador y de las E/S
⑥ - Bornera de tornillos de las salidas
⑦ - Batería, pilas externas de seguridad (TWD LC•• 40DRF únicamente).
⑧ - Resorte de fijación en perfil 35 mm.
⑨ - Alimentación ∿ 100...240 V o ⎓ 24 V
⑩ - Adaptador de cartucho de memoria u marcador de tiempo RTC
⑪ - Puerto Ethernet (TWD LC•• 40DRF)
⑫ - Conector del adaptador de comunicación (puerto serie 2)
⑬ - Potenciómetros analógicos

Figura 2. Descripción hardware de Twido Compacto.

3.2.1.6. Elementos básicos de Twido

Objetos y variables del sistema

La siguiente tabla enumera los objetos y variables del sistema que puede visualizarse y modificarse mediante el monitor de operación, en el mismo orden en el que se accede a ellos.

Objeto	Variable/Atributo	Descripción	Acceso
Entrada	%Ix.y.z	Valor	Lectura/forzado
Salida	%Qx.y.z	Valor	Lectura/escritura/forzado
Temporizador	%TMX.V %TMX.P %TMX.Q	Valor actual Valor preestablecido Hecho	Lectura/escritura Lectura/escritura Leer
Contador	%Cx.V %Cx.P %Cx.D %Cx.E %Cx.F	Valor actual Valor preestablecido Hecho Vacío Completo	Lectura/escritura Lectura/escritura Leer Leer Leer
Bit de memoria	%Mx	Valor	Lectura/escritura
Memoria de palabras	%MWx(3)	Valor	Lectura/escritura
Palabra constante	%KWx	Valor	Leer
Bit de sistema	%Sx	Valor	Lectura/escritura
Palabra de sistema	%SWx(4)	Valor	Lectura/escritura
Entrada analógica	%IWx.y.z	Valor	Leer
Salida analógica	%QWx.y.z	Valor	Lectura/escritura
Contador rápido (FC)	%FCx.V %FCx.VD(1) %FCx.P %FCx.PD(1) %FCx.D	Valor actual Valor actual Valor preestablecido Valor preestablecido Hecho	Leer Leer Lectura/escritura Lectura/escritura Leer

Tabla 1. Listado de objetos y variables de sistema

Direccionamiento de Entradas/Salidas

El direccionamiento es necesario para saber el elemento al cual dirigirnos. Funciona de la siguiente forma:
Para trabajar con Entradas/Salidas

%	I, Q	x	.	y	.	z
Símbolo	Tipo de objeto	Posición del controlador	Punto	Tipo de E/S	Punto	Número de canal

Para trabajar con las palabras de Entradas/Salidas:

%	I, Q	W	x	.	y
Símbolo	Tipo de objeto	Formato	Posición del controlador	Punto	Tipo de E/S

En la tabla siguiente se describe el formato de direccionamiento de E/S:

Grupo	Elemento	Valor	Descripción
Símbolo	%	-	El símbolo de porcentaje siempre precede a una dirección interna.
Tipo de objeto	I	-	Entrada. La "imagen lógica" del estado eléctrico de un controlador o entrada del módulo de E/S de ampliación.
	Q	-	Salida. La "imagen lógica" del estado eléctrico de un controlador o salida del módulo de E/S de ampliación.
Posición del controlador	x	0 1 - 7	Controlador maestro (maestro de conexión remota). Controlador remoto (esclavo de conexión remota).
Tipo de E/S	y	0 1 - 7	Base del módulo de E/S (E/S locales del controlador). Módulos de E/S de ampliación.
Número de canal	z	0 - 31	Número de canal de E/S en el controlador o en el módulo de ampliación de E/S. El número de puntos de E/S disponibles depende del modelo de controlador o del tipo de módulo de E/S de ampliación.

Tabla 2. Direccionamiento de Entradas/Salidas

Ejemplos:

%I0.0.5 ->Punto de entrada número 5 en el controlador base (E/S local).
%Q0.3.4 -> Punto de salida número 4 en el módulo de E/S de ampliación en la dirección 3 para el controlador base (E/S de ampliación).

Para conseguir información detallada acerca de este autómata, consultar el *Catálogo Twido Schneider Electric 2008.*

3.2.2. Descripción del software de programación

Como se ha comentado anteriormente, el PLC Twido se programa con la herramienta de programación Twidosuite.

Introducción a TwidoSuite

Introducción

TwidoSuite es un entorno de desarrollo gráfico, lleno de funciones para crear, configurar y mantener aplicaciones de automatización para los autómatas programables Twido de Schneider Electric. TwidoSuite permite crear programas con distintos tipos de lenguaje, después de transferir la aplicación para que se ejecute en un autómata.

TwidoSuite

TwidoSuite es un programa basado en Windows de 32 bits para un ordenador personal (PC) que se ejecuta en los sistemas operativos Microsoft Windows2000/XP Professional/Vista.

Las principales funciones del software TwidoSuite son:

- Interface de usuario intuitiva y orientada a proyectos.
- Diseño de software sin menús. Las tareas y funciones del paso seleccionado de un proyecto siempre se encuentran visibles.
- Soporte de programación y configuración
- Comunicación con el autómata
- Ayuda de primera mano acerca del nivel de tareas que ofrece enlaces relevantes a la ayuda en línea.

Configuración mínima

La configuración mínima necesaria para utilizar TwidoSuite es la siguiente:

- Se recomienda un equipo compatible con PC y procesador Pentium a 466 MHz o superior, se recomiendan 128 MB de RAM o más de 100 MB de espacio libre en el disco duro.
- Sistema operativo: Windows 2000, Windows XP o Windows Vista:
- Evite el uso de los parches 834707-SP1 (corregido por el parche 890175) y 896358 que producen problemas de visualización en la ayuda en línea.
- Se recomienda Service Pack 2 o superior

3.2.2.1. Lenguajes de programación de Twido

Para crear programas de control Twido pueden utilizarse los siguientes lenguajes de programación:

- *Lenguaje Instruction List o Lista de instrucciones:*

Un programa Lista de instrucciones o IL se compone de una serie de expresiones lógicas escritas como una secuencia de instrucciones booleanas.

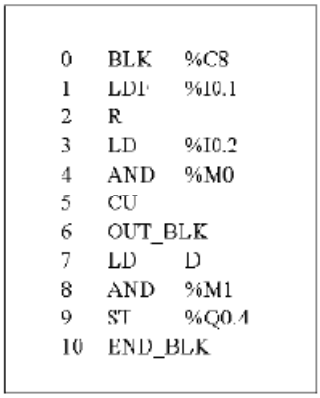


Figura 3. Lista de instrucciones

Lenguaje Instruction List o Lista de instrucciones
Programa escrito en el lenguaje Lista de instrucciones o IL, compuesto por una serie de instrucciones ejecutadas de forma secuencial por el autómata.

- *Ladder Diagrams o Diagramas de contactos:*

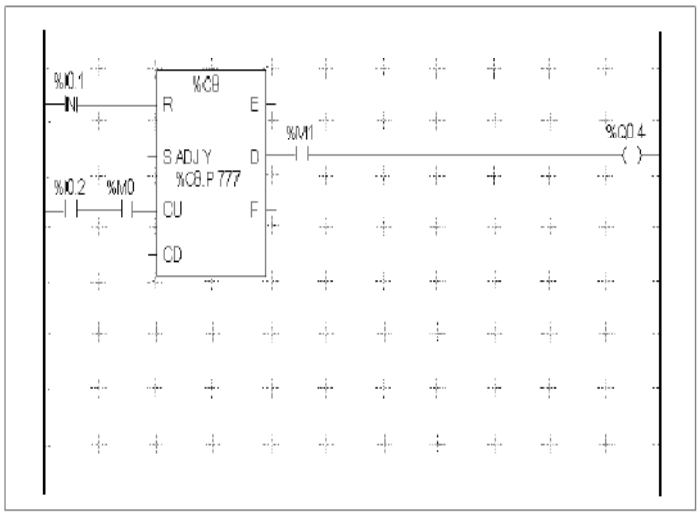


Figura 4. Diagrama de contactos

Un diagrama Ladder es una forma gráfica de mostrar una expresión lógica. Los diagramas de contactos son similares a los diagramas lógicos de relé que representan circuitos de control de relé. En dichos esquemas, los elementos gráficos, como las bobinas, los contactos y los bloques, representan las instrucciones del programa.

- *Lenguaje Grafcet:*

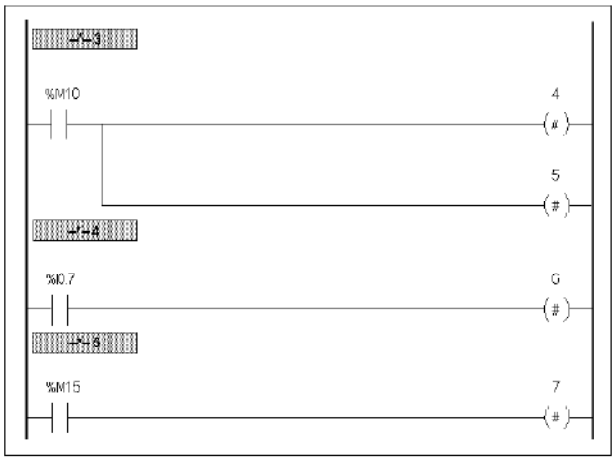


Figura 5. Lenguaje Grafcet

El lenguaje Grafcet está compuesto por una sucesión de pasos y transiciones. Twido admite las instrucciones de lista Grafcet, pero no Grafcet gráfico.
Es un método analítico que divide cualquier sistema de control secuencial en una serie de pasos a los que se asocian acciones, transiciones y condiciones.

La función de reversibilidad de Lista/Ladder Logic permite pasar un programa de Lista a Ladder y viceversa, según convenga.

3.2.2.2. Creación de un proyecto

Para comenzar a programar se debe crear un proyecto nuevo en la pantalla principal de Twidosuite, donde se da nombre y ubicación del proyecto. También se configura la información de proyecto.

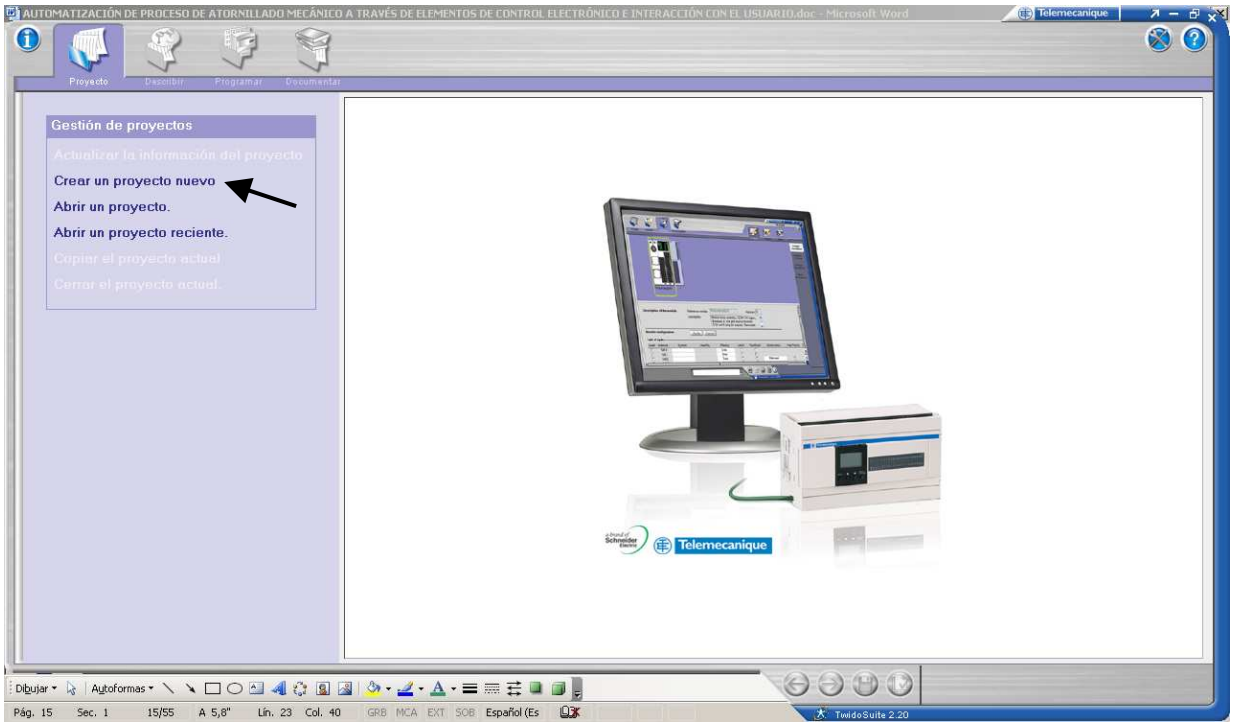


Figura 6. Pantalla principal Twidosuite

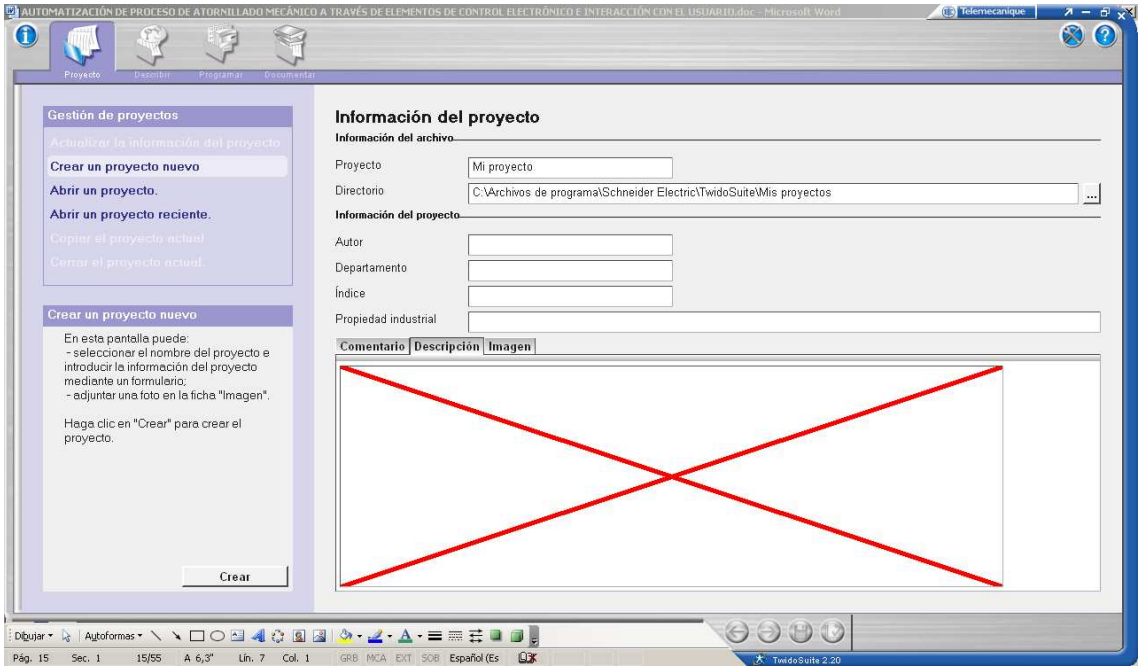


Figura 7. Información de proyecto

Una vez creado el proyecto en Twidosuite, se pasa al siguiente apartado de configuración Hardware, donde las acciones a realizar son:

- Elección CPU a programar
- Configuración de nexo Modbus para el terminal HMI.
- Configuración de nexo Canopen para los encoder.

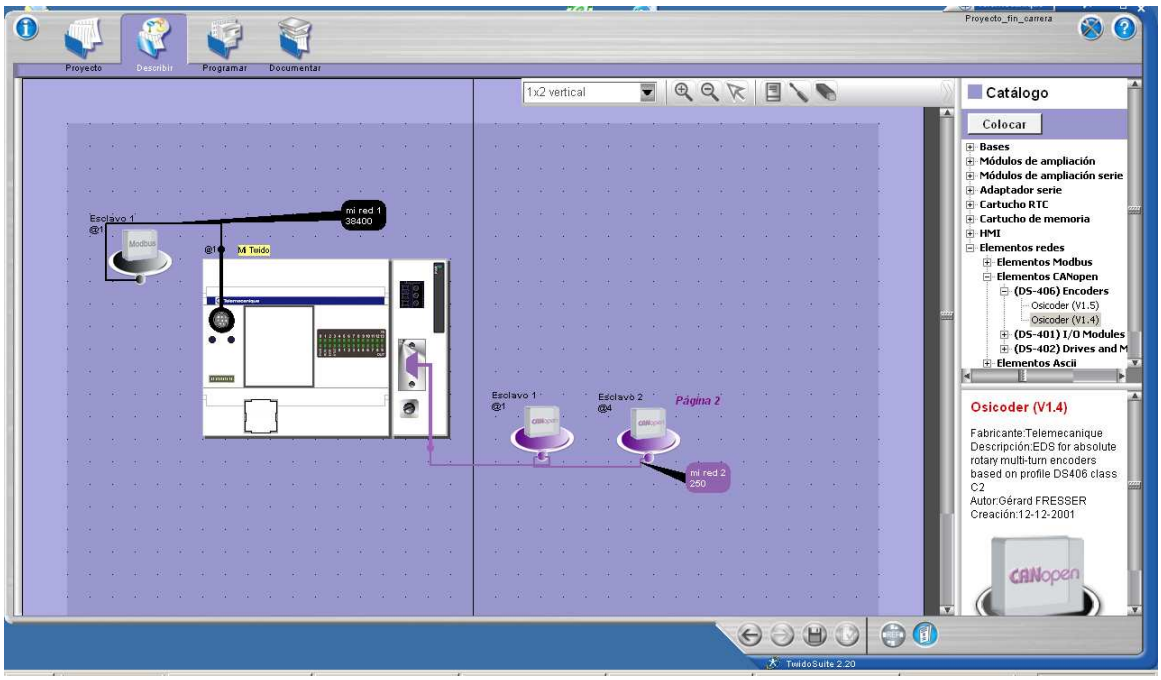
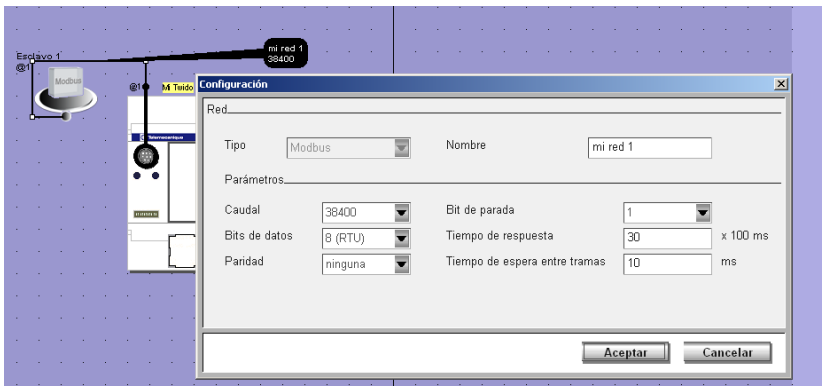


Figura 8. Arquitectura hardware del sistema PLC

Elección CPU a programar

Para la elección de la CPU, se dispone de un catálogo Hardware en la parte derecha de la aplicación. En la sección de bases se selecciona la CPU con la que trabajar y se arrastra al entorno de trabajo.

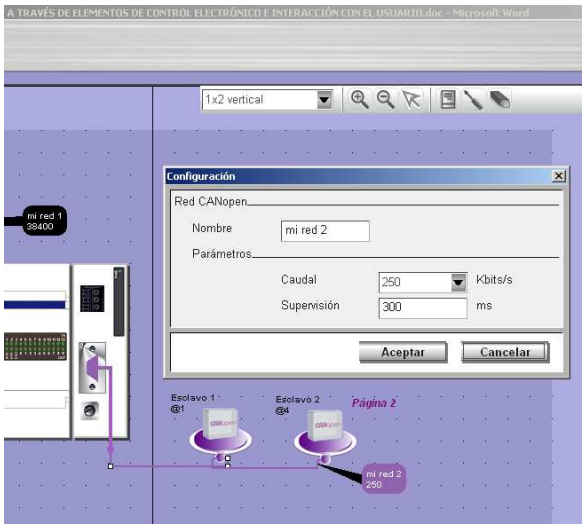
Configuración de Modbus para el nexo PLC-HMI



Se escoge en el catálogo la sección de Elementos Modbus, el dispositivo Magelis HMI. Pulsando sobre la representación de la red Modbus, se determinan los parámetros de comunicación Modbus.

Figura 9. Configuración parámetros red Modbus

Configuración de Canopen para nexo PLC-Encoder



Se escoge en el catálogo la sección de Elementos Canopen, el dispositivo Encoder-Osicoder. Pulsando sobre la representación de la red Canopen, se determinan los parámetros de comunicación Canopen. (velocidad y supervisión). (En caso de no encontrarse los dispositivos se necesitará importar el fichero de configuración Canopen xxxx.eds).

Figura 10. Configuración parámetros Canopen

Configuración de simbólicos de las entradas/salidas digitales

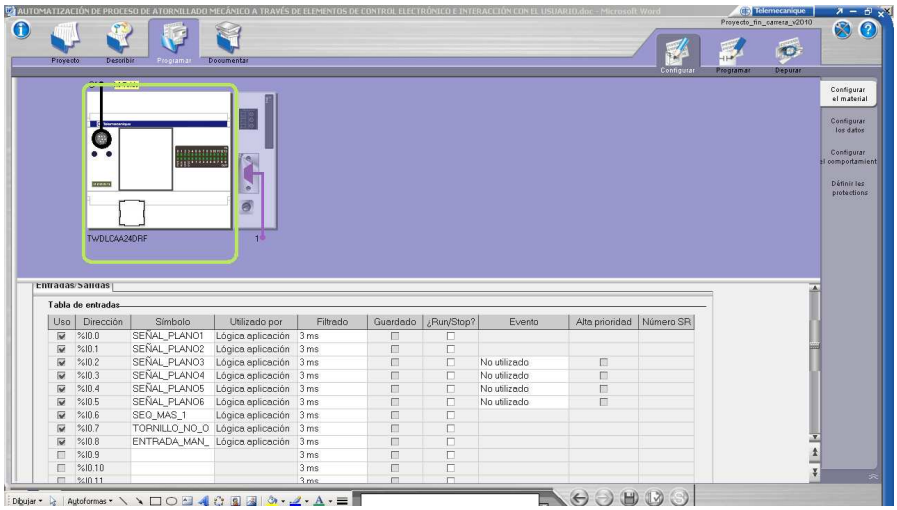


Figura 11. Configuración de simbólicos de las entradas/salidas digitales

Al entrar en la configuración de la CPU se le da nombre simbólico a las entradas/salidas, el cual permitirá aludir a dichas zonas de memoria por su nombre simbólico. Dichas entradas se cablearán posteriormente en el sistema de atornillado para su control. La opción de filtrado permite evitar posibles ruidos en las señales digitales.

Zona programación

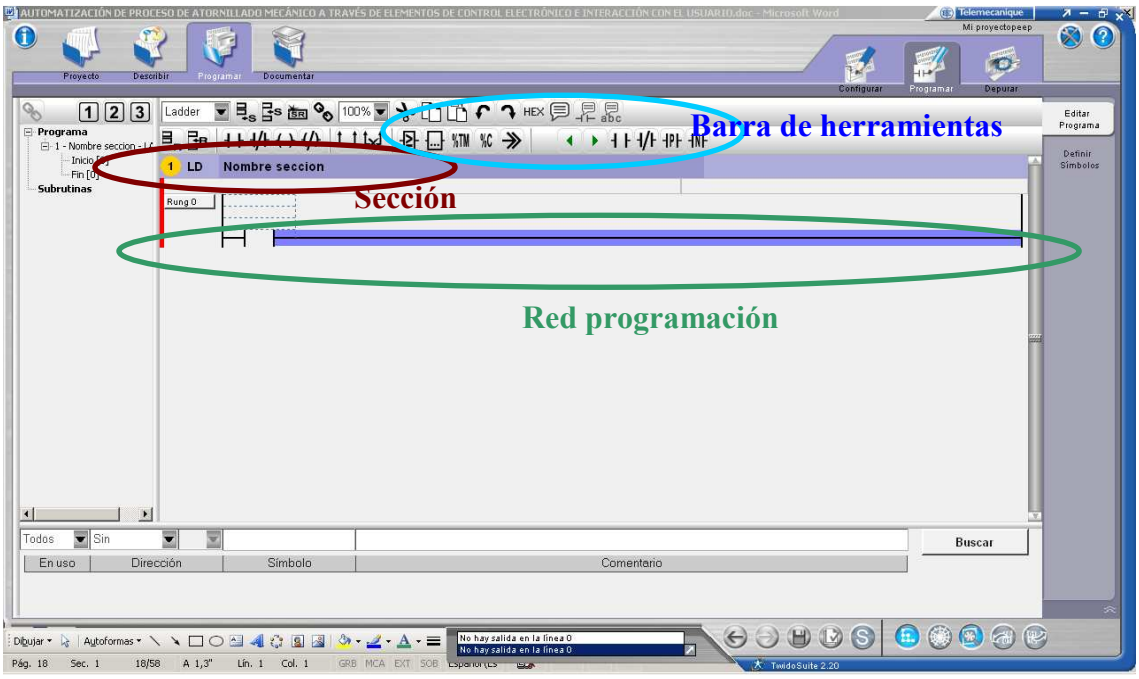


Figura 12. Zona de programación

El siguiente paso consiste en abordar la programación en el espacio de trabajo dedicado a tal fin. El programa Twidosuite divide el programa en secciones, que son las divisiones funcionales del programa, y el redes, las cuales van destinadas a contener las partes lógicas del programa. En el espacio de trabajo se dispone de una barra de herramientas de programación donde se escogerá las herramientas lógicas que se necesite utilizar.

Transferencia programa al PLC

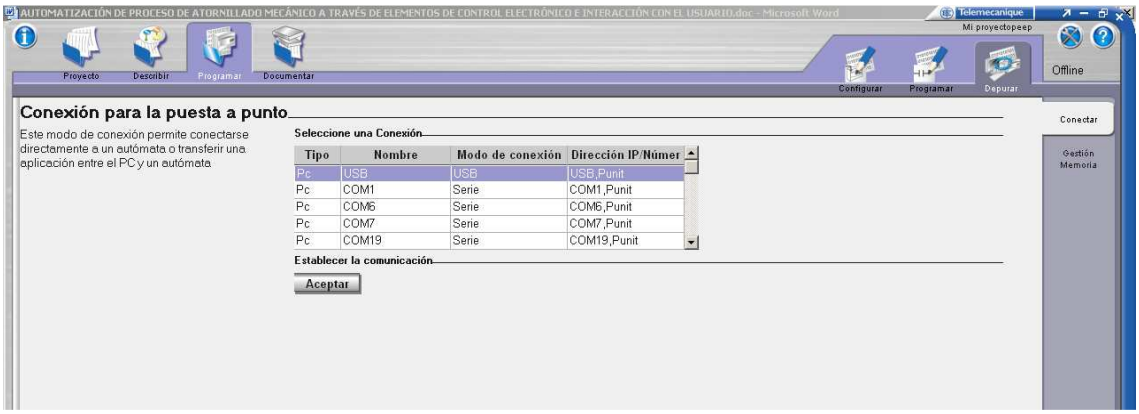


Figura 13. Transferencia programa PLC

Una vez finalizado el programa se finaliza la operación transfiriendo el código compilado al PLC, donde se puede ver su funcionamiento y realizar una posterior depuración de programa.

3.3. Introducción a las redes de comunicación Industriales

Un **bus de campo** es un sistema de transmisión de información (datos) que sustituye normalmente a los sistemas de control e información de cableado físico. Simplifica enormemente la instalación y operación de máquinas y equipamientos industriales utilizados en procesos de producción.

Actualmente, se entienden los buses de campo como elementos imprescindibles como solución en cualquier proyecto de automatización incluso a nivel de máquina. Una de las claras tendencias dentro del campo de la automatización industrial es sin duda el uso cada vez más generalizado de sistemas donde las entradas/salidas se encuentran descentralizadas del cuadro de control. Los argumentos y causas que pueden explicar este hecho son fundamentalmente de tipo económico y originados en la necesidad de las empresas de ser más competitivas día a día. Obviamente, la necesidad de instalar dispositivos de forma remota dentro de una instalación siempre ha estado ahí. Lo que ocurre es que la solución mediante cableado clásico era la única para el fabricante o instalador. El bus de campo representa considerables ahorros en términos de cableado, así como en el coste final de las instalaciones.

El uso de este tipo de buses, según en qué caso, podrá permitir un rápido diagnóstico y, por lo tanto una resolución más efectiva de los problemas que puedan surgir en la instalación. Un punto a tener en cuenta será la estructura modular con la que nos podemos encontrar, ya que esta nos permitirá construir o diseñar sistemas de control desde equipos estandarizados.

3.3.1. Protocolo Modbus

Introducción

La designación Modbus Modicon corresponde a una marca registrada por Gould Inc. Como en tantos otros casos, la designación no corresponde propiamente al estándar de red, incluyendo todos los aspectos desde el nivel físico hasta el de aplicación, sino a un protocolo de enlace (nivel OSI 2). Puede, por tanto, implementarse con diversos tipos de conexión física y cada fabricante suele suministrar un software de aplicación propio, que permite parametrizar sus productos. No obstante, se suele hablar de MODBUS como un estándar de bus de campo, cuyas características esenciales son las que se detallan a continuación.

Estructura de la red

Medio Físico

El medio físico de conexión puede ser un bus semidúplex (half duplex) (RS-485 o fibra óptica) o dúplex (full duplex) (RS-422, BC 0-20mA o fibra óptica).

La comunicación es asíncrona y las velocidades de transmisión previstas van desde los 75 baudios a 19.200 baudios. La máxima distancia entre estaciones depende del nivel físico, pudiendo alcanzar hasta 1200 m sin repetidores.

Acceso al Medio

La estructura lógica es del tipo maestro-esclavo, con acceso al medio controlado por el maestro. El número máximo de estaciones previsto es de 63 esclavos más una estación maestra.
Los intercambios de mensajes pueden ser de dos tipos:

• Intercambios punto a punto, que comportan siempre dos mensajes: una demanda
del maestro y una respuesta del esclavo (puede ser simplemente un reconocimiento («acknowledge»).

• Mensajes difundidos. Estos consisten en una comunicación unidireccional del maestro a todos los esclavos. Este tipo de mensajes no tiene respuesta por parte
de los esclavos y se suelen emplear para mandar datos comunes de configuración,
reset, etc.

Protocolo

La codificación de datos dentro de la trama puede hacerse en modo ASCII o puramente binario, según el estándar RTU (Remote Transmission Unit). En cualquiera de los dos casos, cada mensaje obedece a una trama que contiene cuatro campos principales, según se muestra en la figura 1. La única diferencia estriba en que la trama ASCII incluye un carácter de encabezamiento («:»=3AH) y los caracteres CR y LF al final del mensaje.
Pueden existir también diferencias en la forma de calcular el CRC, puesto que el formato RTU emplea una fórmula polinómica en vez de la simple suma en módulo 16. Con independencia de estos pequeños detalles, a continuación se da una breve descripción de cada uno de los campos del mensaje:

Figura 14. Trama genérica del mensaje según el código empleado

Número de esclavo (1 byte):

Permite direccionar un máximo de 63 esclavos con direcciones que van del 01H hasta 3FH. El número 00H se reserva para los mensajes difundidos.

Código de operación o función (1 byte):

Cada función permite transmitir datos u órdenes al esclavo. Existen dos tipos básicos de órdenes:

• Ordenes de lectura/escritura de datos en los registros o en la memoria del esclavo.

• Ordenes de control del esclavo y el propio sistema de comunicaciones (RUN/STOP, carga y descarga de programas, verificación de contadores de intercambio, etc.)

La tabla 1 muestra la lista de funciones disponibles en el protocolo MODBUS con sus correspondientes códigos de operación.

Campo de subfunciones/datos (n bytes):

Este campo suele contener, en primer lugar, los parámetros necesarios para ejecutar la función indicada por el byte anterior. Estos parámetros podrán ser códigos de subfunciones en el caso de órdenes de control (función 00H) o direcciones del primer bit o byte, número de bits o palabras a leer o escribir, valor del bit o palabra en caso de escritura, etc.

Palabra de control de errores (2 bytes):
En código ASCII, esta palabra es simplemente la suma de comprobación ('*checksum*') del mensaje en módulo 16 expresado en ASCII. En el caso de codificación RTU el CRC se calcula con una fórmula polinómica según el algoritmo mostrado en la figura 15.

Función	Código	Tarea
0	00H	Control de estaciones esclavas
1	01H	Lectura de *n* bits de salida o internos
2	02H	Lectura de *n* bits de entradas
3	03H	Lectura de *n* palabras de salidas o internos
4	04H	Lectura de *n* palabras de entradas
5	05H	Escritura de un bit
6	06H	Escritura de una palabra
7	07H	Lectura rápida de 8 bits
8	08H	Control de contadores de diagnósticos número 1 a 8
9	09H	No utilizado
10	0AH	No utilizado
11	0BH	Control del contador de diagnósticos número 9
12	0CH	No utilizado
13	0DH	No utilizado
14	0EH	No utilizado
15	0FH	Escritura de *n* bits
16	10H	Escritura de *n* palabras

Tabla 3. Funciones básicas Modbus y códigos de operación

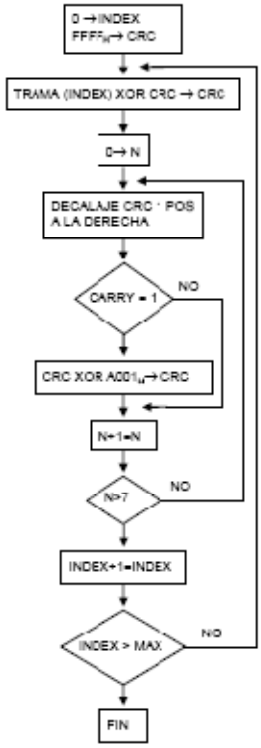


Figura 15. Cálculo del CRC codificación RTU

Descripción de las funciones del protocolo

Función 0

Esta función permite ejecutar órdenes de control, tales como marcha, paro, carga y lectura de programas de usuario del autómata. Para codificar cada una de las citadas órdenes se emplean los cuatro primeros bytes del campo de datos. La trama resultante es la representada en la figura 3 y la interpretación de los códigos de subfunción se especifica en la tabla 2. En caso de las órdenes de marcha y paro, el campo de «información» de la trama representada en la figura 3 está vacío y, por tanto, el mensaje se compone simplemente de 6 bytes de función más 2 bytes de CRC. La respuesta del esclavo a estas órdenes es un mensaje idéntico al enviado por el maestro. Cabe señalar, además, que después de un paro el autómata sólo acepta ejecutar subfunciones de la función 00H.

N° Esclavo (00-3F$_H$)	00$_H$	Código Subfunción SF0 SF1	Datos Subfunción D0 D1	Información	CRC(16) H L

Figura 16. Trama genérica de las subfunciones de control de esclavos (cód. función 00H)

Código subfunción SF0	SF1	Datos subfunción D0	D1	Tarea
00_H	00_H	00_H	00_H	Paro del esclavo sin inicializar
00_H	01_H	00_H	00_H	Marcha del esclavo sin inicializar
00_H	02_H	00_H	00_H	Marcha e inicialización del esclavo
00_H	03_H	00_H	XX_H	Lectura de la secuencia XX de programa de usuario en el esclavo
00_H	04_H	YY_H	XX_H	Carga de una secuencia de programa de usuario en el esclavo Petición: YY = secuencia a cargar, XX= próxima secuencia Respuesta: XX= código error, YY= 00

Figura 17. Subfunciones correspondientes a la función =00H

Funciones 1 y2

Lectura de bits del autómata. La trama es la indicada en la figura 4. La forma de direccionamiento de los bits es a base de dar la dirección de la palabra que los contiene y luego la posición del bit. Obsérvese también que la respuesta es dada siempre en octetos completos.

Petición del maestro

N° Esclavo (00-3F$_H$)	01$_H$ o 02$_H$	Dirección 1er Bit PP PB	N° de Bits NN NN	CRC H L

PPP = Dirección de la palabra (hex), B= Dirección del bit dentro de la palabra 0 a F$_H$.

Respuesta del esclavo

N° Esclavo (00-3F$_H$)	01$_H$ o 02$_H$	N° Octetos leídos NN NN	1er Octeto B7..B0	Otros Octetos Hasta máx. 256	CRC H L

Figura 18. Petición y respuesta de la función: Lectura de bits (01H, 02H)

Funciones 3 y 4

Lectura de palabras del autómata. La trama es la indicada en la figura 5. Obsérvese que la petición indica el número de palabras a leer, mientras que en la respuesta se indica el número de octetos leídos.

Petición del maestro

N° Esclavo (00-3F$_H$)	03$_H$ o 04$_H$	Dirección 1ª Palabra PP PP	N° de Palabras NN NN	CRC H L

PPPP = Dirección de la palabra (hex)

Respuesta del esclavo

N° Esclavo (00-3F$_H$)	03$_H$ o 04$_H$	N° Octetos leídos NN NN	1er Palabra H L	Otras Palabras Hasta máx. 128 H L,H L,HL,...	CRC H L

Figura 19. Petición y respuesta de la función: Lectura de palabras (03H,04H)

Función 5

Escritura de un bit. La trama es la indiada en la figura 6. El direccionamiento del bit se efectúa tal como se ha indicado para las funciones 1 y 2.

Petición del maestro

N° Esclavo (00-3F$_H$)	05$_H$	Dirección Bit PP PB	XX$_H$	00$_H$	CRC H L

PPP = Dirección de la palabra (hex), B= Dirección del bit dentro de la palabra 0 a F$_H$.

Respuesta del esclavo

N° Esclavo (00-3F$_H$)	05$_H$	Dirección Bit PP PB	XX$_H$	00$_H$	CRC H L

XX$_H$ = 00H para bit = 0 y XX$_H$= FF$_H$ para bit = 1

Figura 20. Petición y respuesta de la función: Escritura de un bit (05H)

Función 6

Escritura de una palabra. La trama es la indicada en la figura 7.

Petición del maestro

Nº Esclavo (00-3F$_H$)	06$_H$	Dirección Palabra PP PP	Valor Palabra DD DD	CRC H L

Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	06$_H$	Dirección Palabra PP PP	Nº de Palabras DD DD	CRC H L

Figura 21. Petición y respuesta de la función: Escritura de una palabra (06H)

Función 7

Petición de lectura rápida de un octeto. La trama es la mostrada en la figura 8. Obsérvese que la petición no tiene campo de dirección, esto es debido a que el octeto legible por esta función es fijo en cada esclavo y viene fijado en su configuración.

Petición del maestro

Nº Esclavo (00-3F$_H$)	07$_H$	CRC H L

Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	07$_H$	Valor Octeto DD	CRC H L

Figura 22. Petición y respuesta de la función: Lectura rápida de un octeto (07H)

Función 8

Petición del contenido y control de los 8 primeros contadores de diagnóstico de un esclavo (véase tabla 3). Las tramas de petición y respuesta pueden verse en la figura 9.

Petición del maestro

Nº Esclavo (00-3F$_H$)	08$_H$	Código Subfunción SF0 SF1	Dato Subfunción D0 D1	CRC H L

Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	08$_H$	Código Subfunción SF0 SF1	Valor Contador H L	CRC H L

Figura 23. Petición y respuesta de la función: Control de contadores (08H)

Subfunción N°	Código		Datos D0	D1	Tarea
0	00_H	00_H	XY_H	ZT_H	El esclavo envía el eco XYZT de petición como test.
3	00_H	03_H	ZZ_H	00_H	Modifica el carácter de fin de trama en modo ASCII por ZZ_H
10	00_H	$0A_H$	00_H	00_H	Puesta a cero de los contadores
11	00_H	$0B_H$	00_H	00_H	Lectura del contador 1
12	00_H	$0C_H$	00_H	00_H	Lectura del contador 1
13	00_H	$0D_H$	00_H	00_H	Lectura del contador 1
14	00_H	$0E_H$	00_H	00_H	Lectura del contador 1
15	00_H	$0F_H$	00_H	00_H	Lectura del contador 1
18	00_H	12_H	00_H	00_H	Lectura del contador 1

Tabla 4. Descripción de tarea según subfunción

Función 11

La petición del contenido del contador de diagnóstico número 9, no se realiza por la función 8, sino por la función 11. Las tramas de petición y respuestas son las indicadas por la figura 10.

Petición del maestro

Nº Esclavo (00-3F$_H$)	0B$_H$	CRC H L

Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	0B$_H$	00 00	Valor Contador H L	CRC H L

Figura 24. Petición y respuesta de la función: Contenido contador 9 (0BH)

Función 15

Escritura de bits del autómata. La trama es la indicada en la figura 11. La forma de direccionamiento es análoga a la indicada para las funciones 1 y 2.

Petición maestro

Nº Esclavo (00-3F$_H$)	0F$_H$	Dirección 1er Bit PP PB	Nº de Bits NN NN	Nº de Octetos M	Valor de los bits 8xM valores	CRC H L

Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	0F$_H$	Dirección 1er Bit PP PB	Nº de Bits NN NN	CRC H L

Figura 25. Petición y respuesta: Escritura de bits (0FH)

Función 16

Escritura de palabras del autómata. La trama es la indicada en la figura 12.

Petición maestro

Nº Esclavo (00-3F$_H$)	10F$_H$	Dirección 1ª Palabra PP PP	Nº de Palabras NN NN	Nº de Octetos M	Valor de las palabras HL HL ...	CRC H L

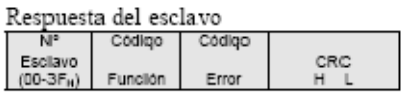
Respuesta del esclavo

Nº Esclavo (00-3F$_H$)	10$_H$	Dirección 1ª Palabra PP PP	Nº de Palabras NN NN	CRC H L

Figura 26. Petición y respuesta: Escritura de palabras (10H)

Mensajes de error

Puede ocurrir que un mensaje se interrumpa antes de terminar. Cada esclavo interpreta que el menaje ha terminado si transcurre un tiempo de silencio equivalente a 3,5 caracteres. Después de este tiempo el esclavo considera que el carácter siguiente es el campo de dirección de esclavo de un nuevo mensaje. Cuando un esclavo recibe una trama incompleta o errónea desde el punto de vista lógico, envía un mensaje de error como respuesta, excepto en el caso de mensajes de difusión. La trama del mensaje de error es la indicada en al figura 13.



Código Función = Código función recibido + 80_H
Código Error = 01 Código de Función erróneo:
02 Dirección incorrecta
03 Datos incorrectos
06 Autómata ocupado

Figura 27. Trama de mensaje de error

Si la estación maestra no recibe respuesta de un esclavo durante un tiempo superior a un límite establecido, declara el esclavo fuera de servicio, a pesar de que al cabo de un cierto número de ciclos hace nuevos intentos de conexión.

Nivel de aplicación

Como se ha dicho a nivel general de buses de campo, el nivel de aplicación de MODBUS no está cubierto por un software estándar, sino que cada fabricante suele suministrar programas para controlar su propia red. No obstante, el nivel de concreción en la definición de las funciones permite al usuario la confección de software propio para gestionar cualquier red, incluso con productos de distintos fabricantes.

Variantes de MODBUS

JBUS

JBUS es una designación utilizada por la firma APRIL para un bus propio que presenta gran similitud con MODBUS, con protocolos prácticamente idénticos.
La designación JBUS, de la misma forma que MODBUS, corresponde a un protocolo de enlace más que a una red propiamente dicha. Puede, por tanto, implementarse con cualquiera de las conexiones físicas normalizadas.

Toda la información relativa al protocolo Modbus se encuentra en la página web oficial de la organización Modbus.

Comparación entre JBUS y MODBUS

La arquitectura de la red, el formato general de la trama y muchos de los códigos de función de ambos buses coinciden exactamente. Existen, sin embargo, algunos códigos de función cambiados, otros que presentan ligeras diferencias o funciones añadidas. Como diferencias más relevantes citaremos las siguientes:

• Posee un registro de estado en cada estación que permite un diagnóstico de la estación.

• El número de esclavo para JBUS (1er byte de la trama) permite valores que van del 01H hasta el FFH. Permite, por tanto, direccionar 255 esclavos en vez de 63. El número 00H se reserva igualmente para mensajes difundidos.

• Las funciones disponibles son prácticamente las mismas en ambos protocolos, pero algunos códigos de función (2º byte de la trama) y de las subfunciones no coinciden.

Función	Código	Tarea
1	01_H	Lectura de *n* bits de salida o internos
2	02_H	Lectura de *n* bits de entradas
3	03_H	Lectura de *n* palabras de salidas o internos
4	04_H	Lectura de *n* palabras de entradas
5	05_H	Escritura de un bit
6	06_H	Escritura de una palabra
7	07_H	Lectura rápida de 8 bits
15	$0F_H$	Escritura de *n* bits
16	10_H	Escritura de *n* palabras

Tabla 5. Funciones idénticas Modbus Jbus

Bus CAN y CAN open

El bus de campo CAN open está basado en el Bus CAN, el cual solo define las capas física y de enlace del bus, por lo que fue necesario definir cómo se asignan y utilizan los identificadores y datos de los mensajes CAN. Para ello se definió el protocolo CAN open, que implementa la capa de aplicación. Actualmente está ampliamente extendido, y ha sido adoptado como un estándar internacional. La construcción de sistemas basados en CAN que garanticen la interoperabilidad entre dispositivos de diferentes fabricantes requiere una capa de aplicación y unos perfiles que estandaricen la comunicación en el sistema, la funcionalidad de los dispositivos y la administración del sistema:

 Capa de aplicación (*application layer*). Proporciona un conjunto de servicios y protocolos para los dispositivos de la red.

 Perfil de comunicación (*communication profile*). Define cómo configurar los dispositivos y los datos, y la forma de intercambiarlos entre ellos.

 Perfiles de dispositivos (*device profiles*). Añade funcionalidad específica a los dispositivos.

La relación entre el modelo OSI y los estándares CAN y CANopen la podemos ver en la figura 1:

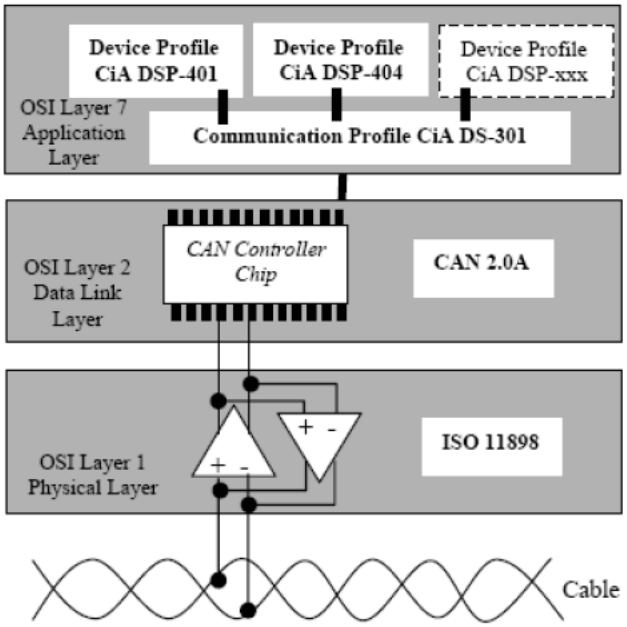


Figura 28. Visión esquemática de los estándares CAN y CANopen en el modelo OSI

CAL (CAN Application Layer)

Fue una de las primeras especificaciones producidas por CiA (CAN in Automation), basada en un protocolo existente desarrollado originalmente por Philips Medical Systems. Ofrece un ambiente orientado a objetos para el desarrollo de aplicaciones distribuidas de cualquier tipo, basadas en CAN.
Esta capa está compuesta por los siguientes cuatro servicios:

- CMS (*CAN-based Message Specification*): ofrece objetos de tipo variable,
  evento y dominio para diseñar y especificar cómo se accede a la funcionalidad
  de un dispositivo través de su interfaz CAN.
- NMT (*Network ManagemenT*): proporciona servicios para la gestión de la red. Realiza las tareas de inicializar, arrancar, parar o detectar fallos en los nodos. Se basa en el concepto de maestro-esclavo, habiendo sólo un NMT maestro en la red.

- DBT (*DistriBuTor*): se encarga de asignar de forma dinámica los identificadores CAN (de 11 bits), también llamados COB-ID (*Communication*
- *Object Identifier*). También se basa en el modelo maestro-esclavo, existiendo sólo un DBT maestro.
- LMT (*Layer ManagemenT*): permite cambiar ciertos parámetros de las capas como por ejemplo el identificador de un nodo (Node-ID) o la velocidad del bus CAN. CMS define 8 niveles de prioridad en sus mensajes, cada uno con 220 COB-IDs, ocupando desde el 1 al 1760. Los identificadores restantes (0, 1761-2031) se reservan para NMT, DBT y LMT, como muestra la Tabla 1: Distribución de los COB-IDs en CAL. En una red CAN el mensaje con el COB-ID más pequeño es el de mayor prioridad. Este estándar asume CAN2.0A (*CAN Standard Message Frame*) con identificadores de 11 bits. Sin embargo se puede utilizar CAN2.0B (*CAN Extended Message Frame*) con identificadores de 29 bits. En ese caso se mapean los 11 bits definidos anteriormente con los 11 bits más significativos del identificador de tramas extendidas. De esta manera los rangos de la tabla quedan bastante más amplios.

COB-ID	Utilización	Cantidad
0	NMT Start/Stop	1
1-220	Objetos CMS Prioridad 0	220
221-440	Objetos CMS Prioridad 1	220
441-660	Objetos CMS Prioridad 2	220
661-880	Objetos CMS Prioridad 3	220
881-1100	Objetos CMS Prioridad 4	220
1101-1320	Objetos CMS Prioridad 5	220
1321-1540	Objetos CMS Prioridad 6	220
1541-1760	Objetos CMS Prioridad 7	220
1761-2015	Monitoreo dispositivos NMT	255
2016-2031	Servicios de NMT, LMT y DBT	16

Tabla 6. Distribución de los COB-IDs en CAL

CANopen

CAL proporciona todos los servicios de gestión de red y mensajes del protocolo pero no define los contenidos de los objetos CMS ni los tipos de objetos. Es decir, define el continente. Aquí es donde el protocolo CANopen añade la funcionaloidad de protocolo restante.
CANopen está por encima de CAL y utiliza un subconjunto de sus servicios y protocolos de comunicación. Proporciona una implementación de un sistema de control distribuido utilizando los servicios y protocolos de CAL.
El concepto central de CANopen es el diccionario de objetos (OD, *Device Object Dictionary*). Es un concepto utilizado por otros buses de campo. No forma parte de CAL.

Diccionario de objetos de CANopen

Es un grupo ordenado de objetos. Describe completamente y de forma estandarizada la funcionalidad de cada dispositivo y permite su configuración mediante mensajes (SDO) a través del propio bus.
Cada objeto se direcciona utilizando un índice de 16 bits. Para permitir el acceso a elementos individuales de las estructuras de datos también existe un subíndice de 8 bits. En la siguiente tabla podemos ver la estructura general del diccionario:

Índice	Objeto	Descripción
0x0000	No usado	
0x0001 – 0x001F	Tipos de dato estáticos	Contiene definiciones de tipos de dato estándar como boolean, enteros, string, punto flotante, etc. Se incluyen como referencia, no pueden ser leídas ni escritas.
0x0020 – 0x003F	Tipos de dato complejos	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, comunes a todos los dispositivos.
0x0040 – 0x005F	Tipos de dato específicos del fabricante	Contiene definiciones de estructuras predefinidas, compuestas de tipos estáticos, específicas de un dispositivo en particular.
0x0060 – 0x007F	Tipos de dato estáticos específicos del perfil del dispositivo	Definiciones de tipos de dato básicos específicos para el perfil del dispositivo.
0x0080 – 0x009F	Tipos de dato complejos específicos del perfil del dispositivo	Definiciones de estructuras específicas para el perfil del dispositivo.
0x00A0 – 0x0FFF	Reservado	
0x1000 – 0x1FFF	Rango para el perfil de comunicaciones	Contiene parámetros de configuración del bus CAN. Estas entradas del diccionario son comunes a todos los dispositivos.
0x2000 – 0x5FFF	Rango para el perfil específico del fabricante	Contiene las extensiones al perfil estándar, realizadas por el fabricante.
0x6000 – 0x9FFF	Rango para perfiles de dispositivo estandarizados	Contiene todos los objetos de datos comunes a un tipo de perfil que pueden ser leídos o escritos desde la red. Algunas de las entradas son obligatorias (funcionalidad requerida) mientras que otras son opcionales (funcionalidad opcional).
0xA000 – 0xFFFF	Reservado	

Tabla 7. Estructura de un diccionario de objetos estándar en CANopen

El rango relevante de objetos va desde el índice 1000 al 9FFF. Para cada nodo de la red existe un OD, diccionario de objetos, que contiene todo los parámetros que describen el dispositivo y su comportamiento en la red.
En CANopen hay documentos que describen perfiles. Hay un perfil de comunicaciones (*communication profile*) donde están descritos todos los parámetros relacionados con las comunicaciones. Además hay varios perfiles de dispositivos (*device profiles*) donde se definen los objetos de un dispositivo en particular.

Un perfil define para cada objeto del diccionario su función, nombre, índice, subíndice, tipos de datos, si es obligatorio u opcional, si es de tipo "sólo lectura", "sólo escritura" o "lectura-escritura".

Identificadores de mensaje

CANopen define la distribución de los identificadores de mensaje (*Predefined Connection Set*) de manera que hay un mensaje de emergencia por nodo, mensajes de sincronización y *time stamp*, un SDO (ocupando dos identificadores), mensajes NMT y cuatro PDOs de transmisión y cuatro de recepción por dispositivo. El identificador de 11 bits se divide en dos partes:

- 4 bits para el código de función
- 7 bits para el identificador de nodo (Node-ID)

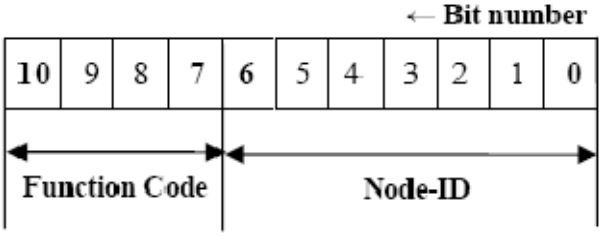


Figura 29. Estructura del identificador de mensajes CAN

La distribución de los identificadores se corresponde con una estructura del tipo maestro-esclavo. El maestro que conoce los Node-IDs de todos los esclavos conectados (máximo 127). Dos esclavos no pueden comunicarse entre sí porque no conocen sus identificadores.

En las siguientes tablas se puede ver la distribución general de los identificadores:

Broadcast objects of the CANopen Predefined Master/Slave Connection Set			
Object	Function code (ID-bits 10-7)	COB-ID	Communication parameters at OD index
NMT Module Control	0000	000h	–
SYNC	0001	080h	1005h, 1006h, 1007h
TIME STAMP	0010	100h	1012h, 1013h

Peer-to-Peer objects of the CANopen Predefined Master/Slave Connection Set			
Object	Function code (ID-bits 10-7)	COB-ID	Communication parameters at OD index
EMERGENCY	0001	081h - 0FFh	1024h, 1015h
PDO 1 (transmit)	0011	181h - 1FFh	1800h
PDO 1 (receive)	0100	201h - 27Fh	1400h
PDO 2 (transmit)	0101	281h - 2FFh	1801h
PDO 2 (receive)	0110	301h - 37Fh	1401h
PDO 3 (transmit)	0111	381h - 3FFh	1802h
PDO 3 (receive)	1000	401h - 47Fh	1402h
PDO 4 (transmit)	1001	481h - 4FFh	1803h
PDO 4 (receive)	1010	501h - 57Fh	1403h
SDO (transmit/server)	1011	581h - 5FFh	1200h
SDO (receive/client)	1100	601h - 67Fh	1200h
NMT Error Control	1110	701h - 77Fh	1016h, 1017h

Tabla 8. Asignación de los identificadores CAN en CANopen

Modelo de comunicaciones

El modelo de comunicaciones de CANopen define cuatro tipos de mensajes (objetos de comunicación):

- Objetos administrativos: son mensajes administrativos que permiten la configuración de las distintas capas de la red así como la inicialización, configuración y supervisión de la misma. Se basa en los servicios NMT, LMS (LSS) y DBT de la capa CAL.

- *Service Data Objects* (SDO): objetos o mensajes de servicio utilizados para leer y escribir cualquiera de las entradas del diccionario de objetos de un dispositivo. Corresponden a mensajes CAN de baja prioridad.

- *Process Data Objects* (PDO): objetos o mensajes de proceso utilizados para el intercambio de datos de proceso, es decir, datos de tiempo real. Por este motivo, típicamente corresponden a mensajes CAN de alta prioridad.

- Mensajes predefinidos: de sincronización, de emergencia y *time stamp*. Permiten la sincronización de los dispositivos (objetos SYNC) y generar notificaciones de emergencia en forma opcional.

CANopen soporta los modelos de comunicación punto-a-punto, maestro-esclavo y productor-consumidor en sus variantes *push* y *pull*. En el modelo *push* los productores colocan los eventos en el canal de eventos y éste se los envía a los consumidores. En el *pull* el flujo de eventos ocurre en el sentido contrario, es decir, los consumidores solicitan eventos al canal de eventos y éste los solicita a los productores.

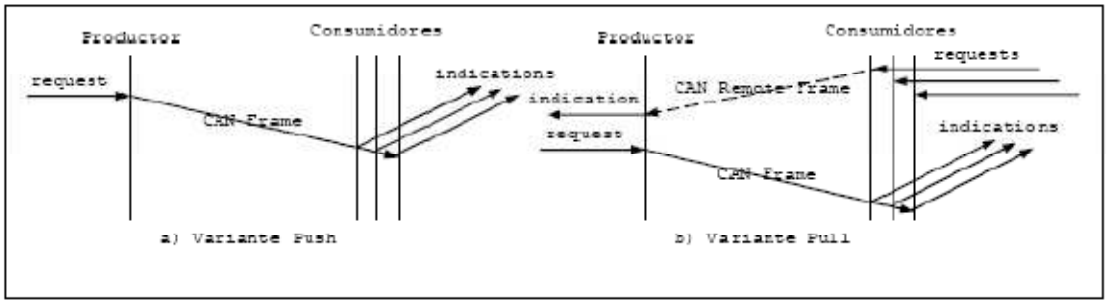


Figura 30. Modelo de comunicación productor-consumidor en CANopen

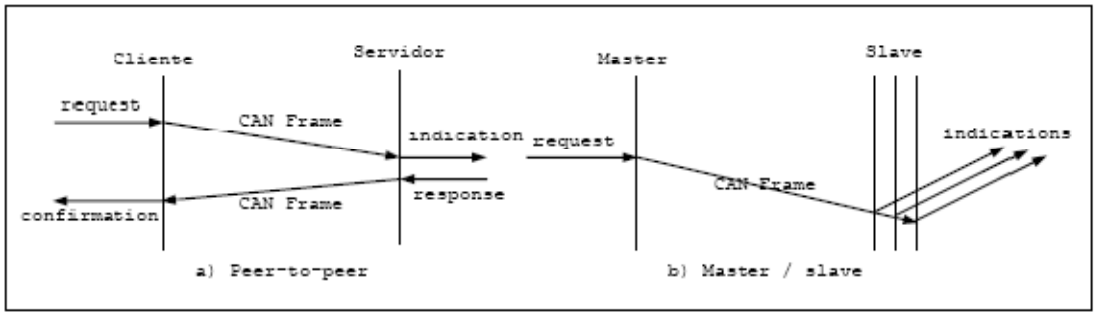


Figura 31. Modelos de comunicación punto-a-punto y maestro-esclavo en CANopen

Service Data Objects (SDO)

Normalmente este tipo de objetos es usado para configuración de dispositivos y transferencia de grandes cantidades de datos no relevantes en forma directa para el control del proceso. En comparación con los PDOs, son mensajes de baja prioridad. Los objetos de servicio o SDOs permiten implementar los modelos de comunicación cliente-servidor o punto-a-punto, para acceder a los diccionarios de objetos de los dispositivos. Para un cierto SDO, el dispositivo cuyo diccionario está siendo accedido es el servidor mientras que el otro dispositivo, el que inicia la actividad, es el cliente. Este tipo de objetos ofrece transferencia de datos sin conexión y con confirmación. Por este motivo, cada SDO involucra el intercambio de dos tramas CAN con diferentes identificadores. Un SDO es representado en CMS como un objeto de tipo *Multiplexed Domain*.

Se definen una serie de protocolos petición-respuesta que se pueden aplicar a los SDOs para su transferencia:

- *Initiate Domain Download*
- *Initiate Domain Upload*
- *Download Domain Segment*
- *Upload Domain Segment*
- *Abort Domain Transfer*

Download significa escribir en el diccionario de objetos y *upload* leer del él.
Al ser *Multiplexed Domains* desde el punto de vista de CMS, los SDOs pueden transferir datos de cualquier longitud. Sin embargo CANopen define dos tipos de transferencia para los SDO basándose en el tamaño de los datos a transferir:

transferencia expédita, para datos de longitud menor o igual a 4 *bytes*
transferencia en segmentos, para datos de longitud mayor de 4 *bytes*.

Los mensajes tanto del cliente como del servidor siempre tienen una longitud de 8 *bytes* aunque no todos contengan información significativa.
Es importante tener en cuenta que en CANopen los parámetros de más de un *byte* se envían siempre en la forma *little endian*, es decir, primero el *byte* menos significativo (LSB).

Transferencia expédita

Usada para transmitir mensajes con longitud de datos menor o igual a 4 *bytes*, para lo cual se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload*. No se aplica fragmentación, se envía un único mensaje CAN y se recibe la confirmación del servidor. Un SDO que se transmite del cliente al servidor bajo este protocolo tiene el siguiente formato:

- ***Command Specifier* (1 *byte*):** contiene información sobre si es subida o descarga de datos, petición o respuesta, transferencia expédita o en segmentos, tamaño de los datos y el *toggle* bit:

  - *Client Command Specifier* (3 bits): 001 para *download* y 010 para *upload*.
  - Ignorado (1 bit): se pone a 0.
  - Número de *bytes* / Ignorado (2 bits): número de *bytes* que no contienen datos. Es válido si los siguientes dos bits son 1. Si no, vale 0. En *upload* se ignora.
  - *Transfer Expedited* / Ignorado (1 bit): indica si se trata de una transferencia expédita (a 1) o una transferencia en segmentos (a 0). Tiene que estar siempre a 1 para este tipo de transferencia. En *upload* se ignora.
  - *Block Size Indicated* / Ignorado (1 bit): si está a 1 es que el tamaño de los datos está especificado, si no, no. En *upload* se ignora.

- ***Index* (2 *bytes*):** índice de la entrada del diccionario de objetos del servidor que el cliente desea acceder mediante el SDO actual. Este campo y el siguiente forman el multiplexor del dominio.

- ***Subindex* (1 *byte*):** subíndice de la entrada del diccionario de objetos del servidor que el cliente desea acceder. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.

- **Datos / Ignorado (4 *bytes***): datos que se desean enviar al servidor (el valor de una entrada en el diccionario si se está escribiendo). Si es *upload* se ignora.

El servidor, al recibir el mensaje anterior, y si ha accedido con éxito al diccionario, contesta con un mensaje con el siguiente formato:

- ***Command Specifier* (1 *byte*):**

  - *Server Command Specifier* (3 bits): 011 para *download* y 010 para *upload*.
  - Ignorado (1 bit): se pone a 0.
  - Ignorado / Número de *bytes* (2 bits): si es un *download* se ignora.
  - Ignorado / *Transfer Expedited* (1 bit): si es un *download* se ignora.
  - Ignorado / *Block Size Indicated* (1 bit): si es un *download* se ignora.

- **Ignorado / Datos (4 *bytes***): si es un *download* (escritura) se ignora ya que es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

Transferencia en segmentos

Usada para transmitir mensajes con longitud de datos mayor de 4 *bytes*. Se aplica fragmentación en segmentos partiendo los datos en múltiples mensajes CAN. El cliente espera confirmación del servidor por cada segmento.
Para el primer mensaje tanto del cliente como del servidor, se usan los protocolos *Initiate Domain Download* o *Initiate Domain Upload* según corresponda, con el bit *Transfer Expedited* a 0 (para transferencia en segmentos). Si ese bit está a 0 y el siguiente (*Block Size Indicated*)
está a 1, significa que el campo de datos (de 4 *bytes*) contiene el número de *bytes* que se van a transmitir (al ser *little endian* el *byte* 4 del mensaje contiene el LSB y el 7 el MSB).

CANopen 15

Para los mensajes siguientes se usan los protocolos *Download Domain Segment* o *Upload Domain Segment*, según se quiera leer o escribir una entrada en el diccionario. Los mensajes CAN bajo estos dos protocolos tienen el siguiente formato:

- ***Command Specifier (1 byte):***

  - *Client Command Specifier* (3 bits): 000 para *download* y 011 para *upload*.
  - *Toggle Bit* (1 bit): es un bit que vale 0 o 1 de forma alternada en segmentos consecutivos. La primera vez vale 0. Teniendo en cuenta que el mecanismo de intercambio sólo permite un mensaje pendiente de confirmación, un único bit es suficiente para esto. El servidor se limita a hacer un eco del valor recibido.
  - Número de *bytes* / Ignorado (3 bits): indica el número de *bytes* que no contienen datos, o cero si no especifica el tamaño. Se ignora en el *upload*.
  - *Last Segment Indication* / Ignorado (1 bit): vale 1 si se trata del último segmento del SDO que se está transmitiendo y 0 si hay más segmentos. Para *upload* se ignora.

- ***Datos / Ignorado (7 bytes***):

  Datos que se desean enviar al servidor. Son los *bytes* que no caben en el mensaje CAN inicial. Si es *upload* se ignora. El servidor, al recibir el mensaje anterior, contesta con un mensaje con el formato:

- ***Command Specifier* (1 *byte*):**

  - *Client Command Specifier* (3 bits): 001 para *download* y 000 para *upload*.
  - *Toggle Bit* (1 bit): igual que antes.
  - Ignorado / Número de *bytes* (3 bits): se ignora en el *download*.
  - Ignorado / *Last Segment Indication* (1 bit): se ignora en el *download*.

- ***Ignorado / Datos (7 bytes):*** si es un *download* (escritura) se ignora porque es una confirmación. Si es una lectura o *upload*, contiene el valor leído del diccionario de objetos del servidor.

Abort Domain Transfer

Uno de los protocolos antes mencionados y que aún no se ha explicado es el *Abort DomainTransfer*. Existe la posibilidad que al acceder a las entradas del diccionario de objetos del servidor, se produzca un error. Este protocolo es usado en esos casos para notificar tanto a clientes como a servidores. El formato de los mensajes de este protocolo es el siguiente:

- ***Command Specifier (1 byte):***
    - *Command Specifier* (3 bits): 100 para *Abort Domain Transfer*.
    - Ignorado (5 bits): se ignoran.
- ***Index (2 bytes):*** índice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Este campo y el siguiente forman el multiplexor del dominio.
- ***Subindex* (1 *byte*):** subíndice de la entrada del diccionario de objetos del servidor que causó el error que se está notificando. Sólo tiene sentido si la entrada es de un tipo complejo. Si se trata de una entrada con tipo estático, debe ser 0.
- ***Código de error (4 bytes):*** código que identifica el error.

Los SDOs requieren ser definidos en el diccionario de objetos mediante una estructura que contiene parámetros relacionados con la transmisión de los mismos. La estructura para el primer SDO para servidores tiene un índice de 0x1200 mientras que el primer SDO para clientes, se ubica en la entrada 0x1280. En total, en una red CANopen, se pueden definir hasta 128 SDOs para clientes y 128 SDOs para servidores.

Process Data Objects (PDO)

Este tipo de objetos permite intercambiar datos del proceso en tiempo real. Implementa el modelo de comunicaciones productor-consumidor. Los datos se transmiten desde un productor a varios consumidores.
Ofrece un servicio de transferencia de datos sin conexión y sin confirmación. No se aplica un protocolo de fragmentación y reensamble de los objetos. Los PDOs están pensados para tráfico de tiempo real de alta prioridad, por lo que es conveniente evitar la sobrecarga que CANopen 15 produciría agregar un protocolo de fragmentación y confirmación como el que se usa en los SDOs.
Los mensajes PDO de un nodo o dispositivo pueden dividirse en dos categorías. Los todo son aquello mensajes con información del proceso que el nodo transmite (por ejemplo la lectura de un sensor). Por otro lado, los rPDO son los mensajes con información del proceso que el nodo escucha (por ejemplo un nodo que controle la apertura de una bomba escuchará el bus en busca de órdenes).

El contenido de un PDO está definido tan sólo por su identificador. Tanto el emisor como el receptor deben conocerlo para poder interpretar su estructura interna. Cada PDO se describe mediante dos objetos del diccionario:

- PDO *Communication Parameter*: contiene el COB-ID que utiliza el PDO, el tipo de transmisión, tiempo de inhibición y temporizador.
- PDO *Mapping Parameter*: contiene una lista de objetos del OD contenidos en la PDO, incluyendo su tamaño en bits. CANopen define varios mecanismos de comunicación para la transmisión de PDOs:

  - **Transmisión asíncrona:**
    - *Eventos:* la transmisión de un mensaje es causada por la ocurrencia de un evento específico definido en el perfil del dispositivo.
    - *Temporizador:* existe un temporizador que cada cierto tiempo cause la transmisión.
    - *Solicitud remota:* la transmisión asincrónica de mensajes PDO puede comenzar al recibir una solicitud remota (trama RTR) enviada por otro dispositivo.

  - **Transmisión sincrónica:** la transmisión sincrónica de mensajes PDO es disparada por la expiración de un período de transmisión,sincronizado mediante la recepción de objetos SYNC. Es decir, cada vez que llega un mensaje SYNC, se abre una ventana de transmisión sincrónica. Los PDOs sincrónicos deben ser enviados dentro de esa ventana. Se distinguen dos modos dentro de este tipo de transmisión:
    - *Modo cíclico:* son mensajes que se transmiten dentro de la ventana abierta por el objeto SYNC. No se transmiten en todas las ventanas sino con cierta periodicidad, especificada por el campo *Transmission Type* del *Communication Parameter* correspondiente.
    - *Modo acíclico*: son mensajes que se transmiten a partir de un evento de la aplicación. Se transmiten dentro de la ventana pero no de forma periódica.

En la siguiente tabla podemos ver los distintos modos de transmisión de PDOs, definidos por el *Transmission Type* (entero de 8 bits) del *Communication Parameter*:

Trans-Mission Type	Condition to trigger PDO (B=both needed, O=one or both)			PDO Transmission
	SYNC*	RTR*	Event*	
0	B	-	B	Sync, acyclic
1-240	O	-	-	Sync, cyclic
241-251	-	-	-	*reserved*
252	B	B	-	Sync, after RTR
253	-	O	-	Async, after RTR
254	-	O	O	Async, manufacturer specific event
255	-	O	O	Async, device profile specific event

Tabla 9. Modos de transmisión de PDO´s en CANopen

Mensajes adicionales

Mensaje deTime Stamp

Este tipo de objetos representan una cantidad absoluta de tiempo en milisegundos desde el 1 de Enero de 1984. Proporciona a los dispositivos un tiempo de referencia común.La etiqueta temporal o *time-stamp* se implementa como una secuencia de 48 bits.

Mensaje de sincronización (SYNC)

En una red CANopen, hay un dispositivo que es el productor de objetos SYNC y una serie de dispositivos consumidores de objetos SYNC. Cuando los consumidores reciben el mensaje del productor, abren su ventana de sincronismo y pueden ejecutar sus tareas sincrónicas.
Este mecanismo permite coherencia temporal y coordinación entre los dispositivos. Por ejemplo, un conjunto de sensores pueden leer las variables del proceso controlado en forma coordinada y obtener así una imagen consistente del mismo.El COB-ID usado por este objeto de comunicación puede encontrarse en la entrada 0x1005 del diccionario. Para garantizar el acceso de estos objetos al bus, debería asignárseles un COBID bajo. El conjunto predefinido de conexiones de CANopen sugiere usar un valor de 128. El campo de datos del mensaje CAN de este objeto se envía vacío.
El comportamiento de estos mensajes es determinado por dos parámetros:

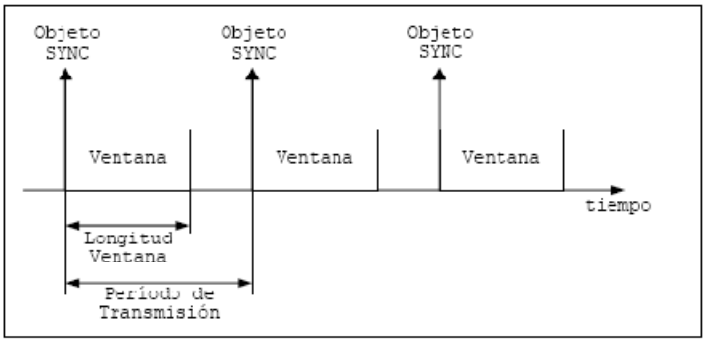


Figura 32. Parámetros de un objeto SYNC en CANopen

Mensaje de emergencia

Estos mensajes se envían cuando ocurre un error interno en un dispositivo. Se transmiten al resto de dispositivos con la mayor prioridad. Pueden usarse como interrupciones o notificaciones de alertas.
Un mensaje de emergencia tiene 8 *bytes*. Su estructura es la siguiente:

COB-ID	Byte 0-1	Byte 2	Byte 3-7
0x080 + Node_ID	**Emergency Error Code**	**Error Register** (Object 0x1001)	Manufac-turer specific error field

Figura 33. Estructura de un mensaje de emergencia en CANopen

- *Emergency Error Code* (2 *bytes*): código del error que causó la generación del mensaje de emergencia. Se trata de fallos internos de los dispositivos por lo cual, los errores se relacionan con fallos de tensión, de corriente, del software del dispositivo, del adaptador del bus CAN, etc.
- *Error Register* (1 *byte*): entrada con índice 0x1001 del diccionario de objetos. Cada bit de este registro indica una condición de error distinta cuando está a '1'.
- *Manufacturer-specific Error Field* (5 *bytes*): este campo puede usarse para información adicional sobre el error. Los datos incluidos y su formato son definidos por el fabricante del dispositivo.

Mensajes de NMT Node/Life Guarding

Se utiliza para saber si un nodo está operativo. La comunicación se basa en el concepto de maestro-esclavo. El NMT maestro monitorea el estado de los nodos. A esto se le llama *node guarding*. Opcionalmente los nodos pueden monitorear el estado del NMT maestro. A esto se le llama *life guarding*. Comienza en el NMT esclavo después de haber recibido el primer mensaje de *node guarding* del NMT maestro. Sirve para detectar errores en las interfaces de red de los dispositivos, pero no fallos en los dispositivos en sí (ya que estos son avisados mediante mensajes de emergencia). Se puede implementar de dos maneras distintas, pudiendo utilizarse sólo una de ellas a la vez: NMT *Node Guarding* o *Heartbeat*.

NMT Node Guarding
El maestro va preguntando a los esclavos si están vivos cada cierto tiempo. Si alguno no contesta en un tiempo determinado significa que está caído y se informa de ello. Si los esclavos también monitorean al maestro deben informar de que el maestro está caído si no reciben mensajes de *Node Guarding* de él durante un determinado intervalo. El maestro interroga a los esclavos mediante una trama remota (RTR) con una estructura y los esclavos responden con un mensaje de confirmación.

Heartbeat
Cada nodo manda un mensaje de *Hearbeat* cada cierto tiempo para informar de que está operativo. En este caso el mensaje de *Boot-up* se considera que es el primer mensaje de *Heartbeat*. Si el NMT maestro deja de recibir estos mensajes durante un tiempo determinado significará que el nodo está caído.

Gestión de la red (NMT)
Aparte de los mensajes predefinidos, CANopen incluye una serie de mensajes para la administración y monitoreo de los dispositivos en la red. Estos están implementados en la capa CAL y reciben el nombre de servicios de gestión de red (*Network ManagemenT*, NMT). Se trabaja con un modelo de comunicaciones maestro-esclavo en el cual un dispositivo es el NMT
maestro y el resto los NMT esclavos. Un dispositivo NMT esclavo puede encontrarse en alguno de los siguientes estados:

- ***Initialising***: al encender el dispositivo se pasa directamente a este estado. Después de realizar las labores de inicialización el nodo transmite el mensaje de *Boot-up* y pasa al estado *Pre-Operational*.

Para permitir un reseteo parcial del nodo se subdivide en tres subestados:

  - *Reset-Application*: los parámetros específicos del fabricante y del perfil del dispositivo son fijados a su valor inicial. A continuación el nodo pasa al estado *Reset-Communication*.
  - *Reset-Communication*: los parámetros del perfil de comunicaciones son fijados a su valor inicial. A continuación el nodo pasa al estado *Initialising*.

- ***Pre-operational***: en este estado el dispositivo puede ser configurado mediante SDOs. Puede enviar y recibir SDOs, mensajes de emergencia, de sincronización, *time stamp* y mensajes NMT.
- ***Operational***: el dispositivo ya ha sido configurado y funciona normalmente. Todos los objetos de comunicación están activos así también puede enviar y recibir PDOs.
- ***Stopped***: todos los objetos de comunicación dejan de estar activos. No se pueden enviar ni recibir PDOs ni SDOs, sólo mensajes de NMT.

Sólo el NMT maestro puede mandar mensajes del módulo de control NMT, pero todos los esclavos deben dan soporte a los servicios del módulo de control NMT.

La descripción de esta comunicación se encuentra de forma detallada en la web oficial de *CAN in Automation (CiA).*

3.4. Sistemas de visualización y supervisión.

3.4.1. Introducción a los elementos HMI táctiles

Sistema de visualización y control para procesos industriales desde terminal gráfico táctil. HMI viene de las siglas de "Human Machine Inteface", es decir: Interfaz Hombre Máquina. Se trata de un elemento especialmente diseñado para establecer los parámetros necesarios de la máquina sin necesidad tener conocimiento de programación de PLC´s o derivados, sino que requiere exclusivamente del conocimiento de la operación de la máquina. Proporciona una interactividad a través de la comunicación con los dispositivos de campo (autómatas programables, etc.) y puede controlar el proceso de forma automática, junto con el PLC. Además, puede proveer de toda la información que se genera en el proceso a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa que requieran contacto con la operación de la máquina, así como control de calidad, supervisión, mantenimiento, etc.
La comunicación se realiza mediante buses de campo (en el caso actual, Modbus RTU y Canopen) o redes LAN. Todo esto se ejecuta normalmente en tiempo real (Runtime), y están diseñados para dar al operador de planta la posibilidad de supervisar y controlar dichos procesos.

Prestaciones

Un terminal gráfico HMI ofrece las siguientes prestaciones:

- Posibilidad de crear paneles de alarma, que exigen la presencia del operador para reconocer una parada o situación de alarma, con registro de incidencias.

- Generación de históricos de señal de planta, que pueden ser volcados para su proceso sobre una hoja de cálculo (con archivos sin formato).

- Ejecución de programas, que modifican la ley de control, o incluso anulan o modifican las tareas asociadas al autómata, bajo ciertas condiciones (siempre que se implemente en el programa del PLC)

- Posibilidad de programación numérica, que permite realizar cálculos aparte de los realizados en el PLC, con lo que podemos distribuir de una forma más adecuada la carga de cálculos a realizar.

Con ellas, se pueden desarrollar aplicaciones para ordenadores (tipo PC, por ejemplo), con captura de datos, análisis de señales, presentaciones en pantalla, envío de resultados a disco e impresora, etc. a través de la carga de Runtime en el PC.
Además, todas estas acciones se llevan a cabo mediante un paquete de funciones que incluye zonas de programación en un lenguaje de uso general (como Java), lo cual confiere una potencia muy elevada y una gran versatilidad.

Características

Un HMI cumple varios objetivos:

- Permiten una fácil e intuitiva interacción con la máquina a través de su capacidad táctil.
- Son elementos fácilmente integrables en arquitecturas abiertas, capaces de contener aplicaciones que pueden crecer o adaptarse según las necesidades cambiantes de la empresa.
- Se comunican con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión).

Pantallas de supervisión Magelis XBTGT.

Para el desarrollo de este proyecto se ha escogido un terminal gráfico XBTGT2220, que cumple con los siguientes requisitos:

- Terminal industrial de marca internacional (Schneider Electric)
- Modo de introducción de variables táctil
- Comunicación con PLC a través de Modbus
- Pantalla color QVGA, pantalla STN
- IEC/EN 61131-2 para controladores programables F
- UL 508 para equipos de control industrial
- UL 1604 para el uso de equipos eléctricos en ubicaciones peligrosas de Clases I y II, División 2, y Clase III.
- UL 60950 para la seguridad de los equipos de tecnología de la información
- CISPR 11 UL50/Nema 250 4X sólo para uso en interiores
- EN 60079-15 y IEC 61241-1

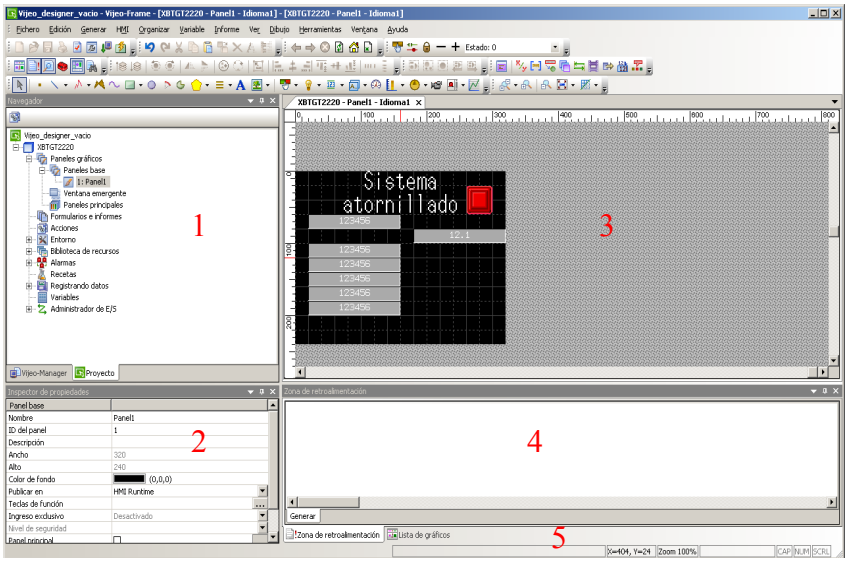
Figura 34. Terminal táctil XBTGT

3.4.2. Programación del terminal táctil

La programación del terminal táctil se hace mediante el software Vijeo Designer de Schneider Electric. El software de configuración Vijeo Designer puede utilizarse para crear aplicaciones de diálogo-operadoor destinadas al control de sistemas de automatismo para:

- Terminales Magelis XBT G y XBT GT de nueva tecnología.
- PC industriales Magelis Smart y Compact iPC.

Vijeo Designer y un terminal adecuado pueden combinarse para ofrecer una soluciónque satisfaga todas y cada una de las necesidades de los fabricantes, por el precio de una simple reconfiguración de software.
Gracias a que admite la producción de imágenes de vídeo, la oferta Magelis Vijeo Designer proporciona acceso a nuevos tipos de aplicaciones. Los usuarios pueden visualizar sus procesos de forma inmediata o en un intervalo de tiempo, en la misma pantalla que el diálogo de HMI.
Vijeo Designer utiliza la conectividad Ethernet TCP/IP de Magelis y admite, por tanto, el acceso remoto WEB Gate, el uso compartido de datos de aplicaciones entre terminales, la transferencia de fórmulas y registros para variables, y mucho más, todo ello con una total seguridad.
Las aplicaciones pueden adoptar una naturaleza internacional, gracias a la capacidad de Vijeo Designer de admitir hasta 10 idiomas simultáneamente en un proyecto (38 alfabetos disponibles en el terminal XBT GT).
El interface y la documentación de Vijeo Designer se ofrecen en 6 idiomas: español, inglés, francés, alemán, italiano y chino simplificado.
Vijeo Designer se ejecutará en cualquier PC con Windows 2000 o Windows XP Professional. Admite la simulación WYSIWYG (se observa el resultado final de la programación del terminal objetivo) de la aplicación ampliada (sin terminal XBT G/GT o Magelis iPC objetivo), la simulación de variables de autómata (E/S, bits y palabras internas), y garantiza que la aplicación se ejecute con total seguridad en el terminal XBT G/GT o Magelis Smart/Compact iPC.

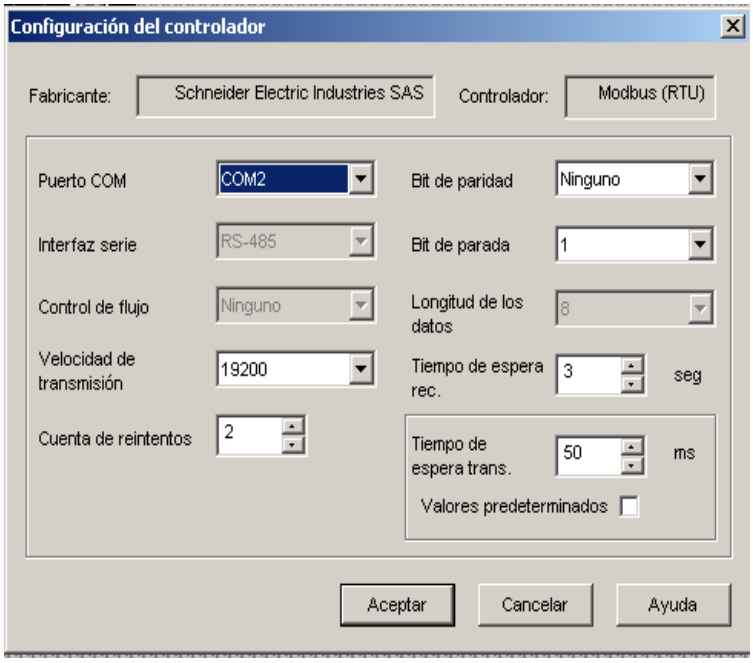


Configuración:
El software de configuración Vijeo Designer permite procesar proyectos de diálogo operador rápida y fácilmente gracias a su avanzada ergonomía que utiliza hasta 5 ventanas configurables:

1.-Ventana de navegador.
2.- Ventana de lista de objetos.
3.- Ventana de fórmulas.
4.- Ventana de biblioteca de objetos gráficos animados y objetos de imágenes.
5.- Ventana de informe.

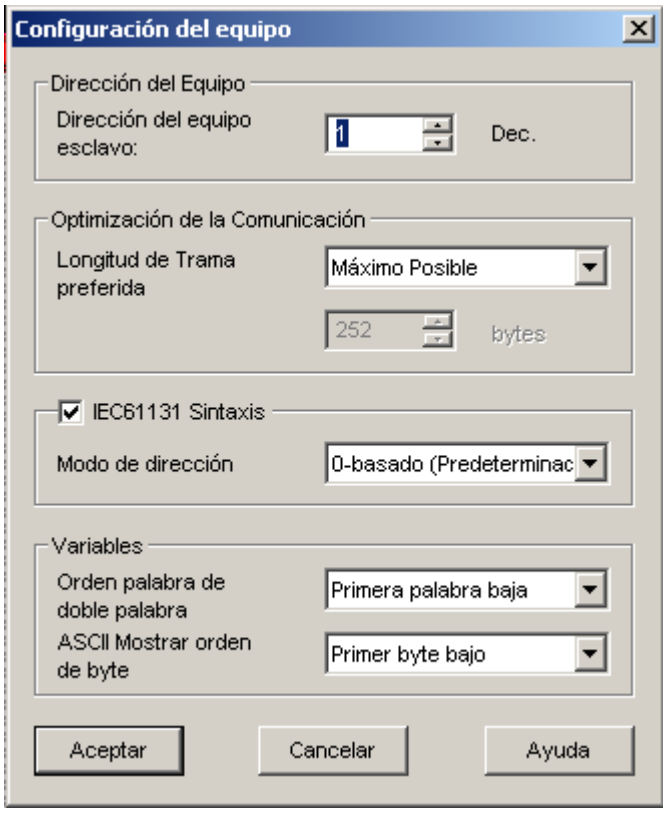
Figura 35. Entorno programación Vijeo Designer

3.4.2.1. Programación de comunicaciones



Las programación del puerto de comunicaciones se hace entrando en la configuración del árbol desplegable de la ventana del navegador "Administrador de E/S", en el apartado COM2. Esta configuración debe coincidir con la programada en el PLC.

Figura 36. Configuración parámetros Modbus



La dirección del equipo esclavo Modbus se hace entrando en la configuración del árbol desplegable de la ventana del navegador "Administrador de E/S", en el apartado Equipo Modbus.

Figura 37. Configuración de dirección del equipo esclavo

El software también ofrece un conjunto completo de herramientas de gestión de aplicaciones para:

- Creación de proyectos, donde un proyecto está constituido por varias aplicacionespara XBT G/XBT GT/Smart iPC/Compact iPC con un uso compartido de variables entre terminales (hasta 8 terminales y 300 variables).
- Gestión de recetas (32 grupos de 256 recetas con un máximo de 1.024 componentes).
- Referencias cruzadas de variables de aplicaciones.
- Documentación de vistas de una aplicación.
- Un modo de simulación que permite probar la aplicación en la oficina de diseño.
- Gestión de lectores de códigos de barras a través de:
  - Puerto USB de terminales XBT GT multifunción y PC industriales Magelis Smart iPC y Compact iPC.
  - Puerto serie COM1 de XBT G, o COM2 de XBT G y XBT GT.

3.4.2.2. Editor gráfico

El editor gráfico de Vijeo Designer ofrece un interface coherente para objetos simples así como para objetos más sofisticados. Permite a los desarrolladores de aplicaciones crear vistas basadas fácilmente en:

- Objetos simples para configurar:
  - Puntos, líneas, rectángulos, elipses, arcos.
  - Gráficos de barras, medidores, depósitos, rellenos, gráficos de sectores, curvas.
  - Polilíneas, polígonos, polígonos regulares, curvas Bézier, escalas.
  - Textos, imágenes o resumen de alarmas, etc.
- Objetos avanzados preconfigurados: interruptores, botones de opción, indicadores, botones, depósitos, gráficos de barras, potenciómetros, selectores, campos de texto o número, listas enumeradas, etc.

3.4.2.3. Animaciones de objetos

La animación de 8 tipos de objetos gráficos permite la creación de vistas animadas basadas en:

- Pulsación del panel táctil.
- Cambio de color.
- Relleno.
- Movimiento.
- Rotación.
- Tamaño.
- Visibilidad.
- Visualización de valor asociado.

3.4.2.4. Java Scripts

Vijeo Designer admite el procesamiento de información con Java Scripts. Esta función facilita la ejecución de animaciones complejas, la automatización de tareas en el terminal y la gestión de cálculos con el fin de reducir la carga de los programas del autómata.
Los Scripts (50 líneas, máx. recomendado) pueden asociarse a:

- Variables.
- Acciones de operador.
- Pantallas.
- La propia aplicación.

3.4.2.5. Funciones avanzadas

Vijeo Designer, basado en las nuevas tecnologías de la información, ofrece un gran número de funciones avanzadas para el procesamiento de un volumen de datos mayorde una forma más rápida y fiable:

- Gestión de datos multimedia en los formatos más conocidos:
  - Visualización de imágenes (archivos jpeg, bmp, emf, y png).
  - Procesamiento y visualización de texto (archivos .txt).
  - Procesamiento de mensajes de sonido (archivos .wav).
- Registros de alarmas o curvas grabados para la transferencia y el almacenamiento de datos.
- Gestión de alarmas. Todas las variables pueden categorizarse como “Alarmas” y pueden personalizarse con respecto a la visualización y el reconocimiento. Estas alarmas analógicas de tipo límite y booleano pueden imprimirse en tiempo real.
- Transferencia de aplicaciones multimodo: a través de un enlace serie Ethernet y una tarjeta de memoria Compact Flash (en terminales multifunción).
- Copia de seguridad de la aplicación en el terminal o iPC para facilitar el mantenimiento.
- Intercambio de datos sencillo entre el PC y el terminal con la herramienta Administrador de Datos.
- Servidor FTP integrado para descargar/cargar a través de Ethernet TCP/IP y restaurar registros en terminales XBT G/GT.
- Pueden activarse simultáneamente una comunicación de múltiples puertos para terminales multifunción, 2 enlaces serie y 1 red Ethernet.

Para ampliar esta información se puede consultar en los *Manuales de curso Vijeo Designer nivel I y nivel II.*

3.5. Sistemas de Captación de posición y velocidad: Encoders

Un encoder es un aparato mecánico que detectan el movimiento mecánico y traducen la información de posición, velocidad y aceleración en datos eléctricos útiles para después poder trabajar con ellos. Existen dos tipos de arquitectura, la lineal y la rotatoria.

3.5.1. Encoder incremental

El encoder es un transductor rotativo que transforma un movimiento angular en una serie de impulsos digitales. Estos impulsos generados pueden ser utilizados para controlar los desplazamientos de tipo angular o de tipo lineal, si se asocian a cremalleras o a husillos. Las señales eléctricas de rotación pueden ser elaboradas mediante controles numéricos (CNC), contadores lógicos programables (PLC), sistemas de control etc. Las aplicaciones principales de estos transductores están en las máquinas herramienta o de elaboración de materiales, en los robots, en los sistemas de motores, en los aparatos de medición y control. En los encoders de producción Schneider Electric, la detección del movimiento angular se ejecuta en base al principio de exploración fotoeléctrica. El sistema de lectura se basa en la rotación de un disco graduado con un reticulado radial formado por líneas opacas, alternadas con espacios transparentes. Este conjunto está iluminado de modo perpendicular por una fuente de rayos infrarrojos. El disco proyecta de este modo su imagen sobre la superficie de varios receptores oportunamente enmascarados por otro reticulado que tiene el mismo paso del anterior llamado colimador. Los receptores tienen la tarea de detectar las variaciones de luz que se producen con el desplazamiento del disco convirtiéndolas en las correspondientes variaciones eléctricas.

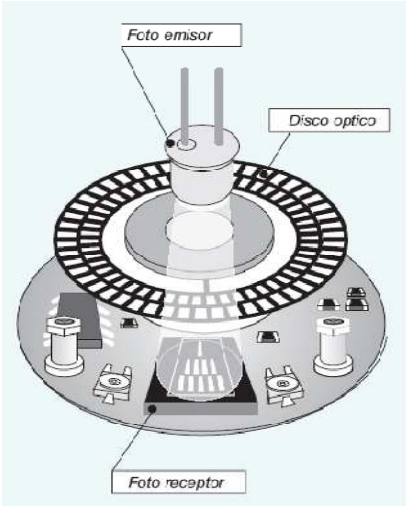


Figura 38. Funcionamiento encoder

La señal eléctrica detectada, para generar impulsos correctamente escuadrados y sin interferencias, debe ser procesada electrónicamente. Para incrementar la calidad y estabilidad de las señales, el sistema de lectura se efectúa generalmente de manera diferencial, comparando dos señales casi idénticas, pero desfasados en 180° eléctricos. Su lectura se efectúa en base a la diferencia de las dos señales, eliminando de este modo las interferencias, porque están superpuestas de igual manera en toda forma de onda.

El encoder incremental proporciona normalmente dos formas de ondas cuadradas y desfasadas entre sí en 90º eléctricos, los cuales por lo general son "canal A" y "canal B". Con la lectura de un solo canal se dispone de la información correspondiente a la velocidad de rotación, mientras que si se capta también la señal "B" es posible discriminar el sentido de rotación en base

a la secuencia de datos que producen ambas señales. Está disponible además otra señal llamado canal Z o Cero, que proporciona la posición absoluta de cero del eje del encoder. Esta señal se presenta bajo la forma de impulso cuadrado con fase y amplitud centrada en el canal A. Representación de las señales incrementales A, B y Z en disco óptico.

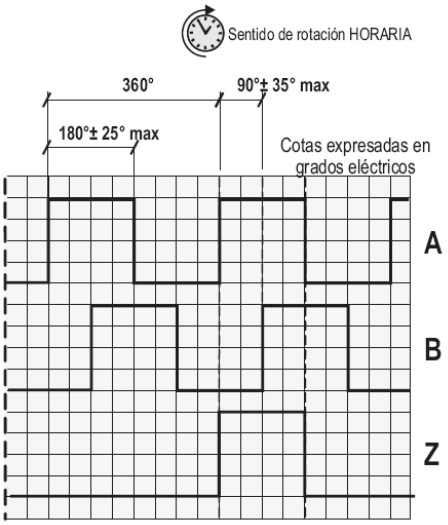


Figura 39. Representación gráfica de las señales A,B, y Z.

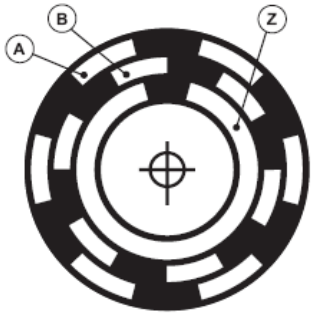


Figura 40. Representación de las señales A,B y Z en disco óptico

La precisión de un encoder incremental depende de factores mecánicos y eléctricos entre los cuales, el error de división del retículo, la excentricidad del disco, la de los rodamientos, el error introducido por la electrónica de lectura, imprecisiones de tipo óptico.

La unidad de medida para definir la precisión de un encoder es el grado eléctrico, éste determina la división de un impulso generado por el encoder: en efecto, los 360º eléctricos corresponden a la rotación mecánica del eje, necesaria para hacer que se realice un ciclo o impulso completo de la señal de salida. Para saber a cuántos grados mecánicos corresponden 360 grados eléctricos es suficiente aplicar la fórmula siguiente.

$$360^{\circ}\,eléctri\cos = \frac{360^{\circ}\,mecáni\cos}{n^{\circ}\,impulso\,/\,giro}$$

El error de división en un encoder, está dado por el máximo desplazamiento expresado en grados eléctricos, de dos frentes de onda consecutivos. Este error existe en cualquier encoder y se debe a los factores antes citados.

3.5.2. Encoder absoluto

Principio de funcionamiento

El principio de funcionamiento de un encoder absoluto es muy similar al de un encoder incremental en el que un disco que gira, con zonas transparentes y opacas interrumpe un haz de luz captado por fotorreceptores, luego éstos transforman los impulsos luminosos en impulsos eléctricos los cuales son tratados y transmitidos por la electrónica de salida.

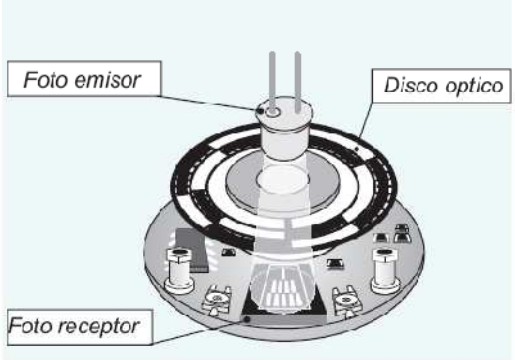


Figura 41. Encoder absoluto

Codificación absoluta

Respecto a los encoders incrementales, los encoders absolutos muestran importantes diferencias desde el punto de vista funcional. Mientras en los encoders incrementales la posición está determinada por el cómputo del número de impulsos con respecto a la marca de cero, en los encoders absolutos la posición queda determinada mediante la lectura del código de salida, el cual es único para cada una de las posiciones dentro de la vuelta. Por consiguiente los encoders absolutos no pierden la posición real cuando se corta la alimentación (incluso en el caso de desplazamientos), hasta un nuevo encendido (gracias a una codificación directa en el disco), la posición está actualizada y disponible sin tener que efectuar, como en el caso de los encoder incrementales la búsqueda del punto de cero.
Analicemos ahora el código de salida que se deberá utilizar para definir la posición absoluta. La elección más obvia es la del código binario, porque fácilmente puede ser manipulado por los dispositivos de control externos para la lectura de la posición, sin tener que efectuar particulares operaciones de conversión. En vista que el código se toma directamente desde el disco (que se encuentra en rotación) la sincronización y la captación de la posición en el momento de la variación entre un código y el otro se vuelve muy problemática. En efecto, si por ejemplo tomamos dos códigos binarios consecutivos como 7(0111) 8(1000), se nota que todos los BIT del código sufren un cambio de estado: una lectura efectuada en el momento de la transición podría resultar completamente errónea porque es imposible pensar que las variaciones sean instantáneas y que se produzcan todas en el mismo momento. Debido a este problema se utiliza una variante del código binario: el código Gray, el cual tiene la particularidad que al pasar entre dos códigos consecutivos (o desde el último código al primero), uno sólo cambia su estado.

DECIMAL	BINARIO	GRAY
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

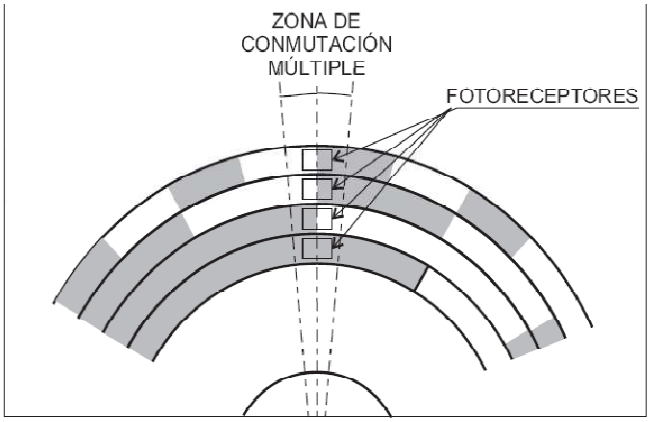


Figura 42. Codificación Decimal, Binario y Gray aplicado en disco óptico.

3.5.3. Selección de los encoder

Los encoder escogidos para el proyecto son de la marca Schneider Electric con referencia **XCC3510PS84CB**, los cuales, tal y como podemos encontrar en el *Catálogo Osicoder de Schneider Electric*, tienen las siguientes características:

- Comunicables en Canopen, protocolo explicado anteriormente, con lo que obtendremos el valor de la posición automáticamente a través de comunicaciones.
- El protocolo de aplicación incluye la programación de las funciones adicionales siguientes:

  - Secuencia del código.
  - Resolución por vuelta.
  - Resolución global.
  - Preselección.
  - Velocidad y dirección.

Las direcciones en el bus Canopen serán 1 y 2 respectivamente. La velocidad de transmisión se configurará a 250 Kbaudios/seg, para conseguir una captura de datos estable.

3.6. Sistema de atornillado automático

3.6.1. Proceso de atornillado

Una de las definiciones de atornillado podría ser el método de unión entre dos o más elementos conseguido a través de apretar un tornillo (Pieza cilíndrica o cónica, por lo general metálica, con resalte en hélice y cabeza apropiada para enroscarla). común a los elementos de unión haciéndolo girar en torno a su eje mediante un atornillador.

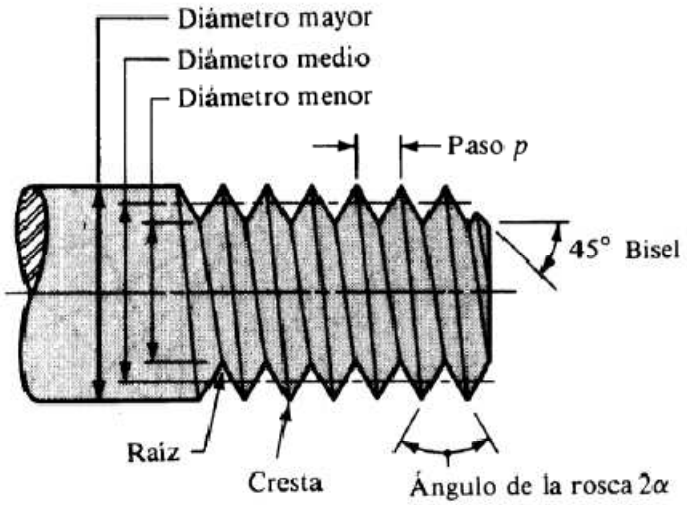
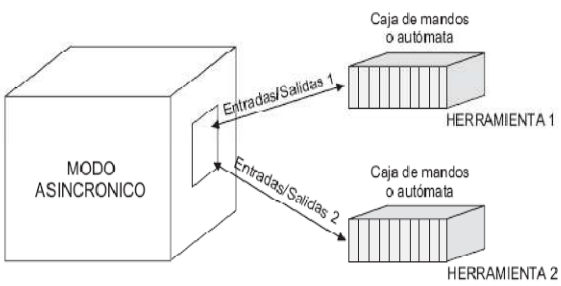


Figura 43. Diagrama de un tornillo

Para ampliar esta información consultar *Elementos de máquinas Ed. Prentice Hall.*
Para desarrollar la función del atornillado automático se ha escogido el atornillador modelo TWINCVI II .

3.6.2. Descripción del sistema de atornillado TWINCVI II

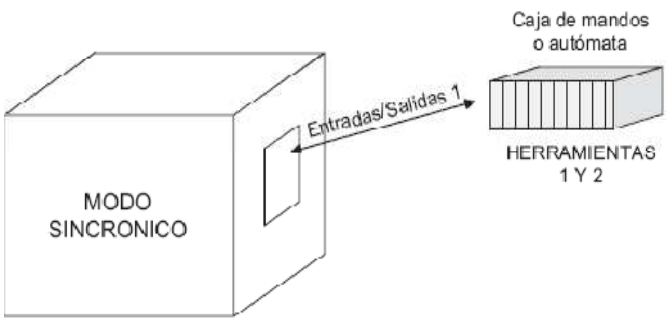
El sistema de atornillado automático permite pilotar 1 ó 2 herramientas de apriete eléctricas portátiles de tipo ER y/o fijas de tipo EM. Se suministra listo para funcionar. La parametrización implícita responde a la mayor parte de las aplicaciones. El TWINCVI II consta de 1 ó 2 máquinas dependiendo de si el modo de funcionamiento es sincrónico o asincrónico.
La elección del modo de funcionamiento se hará en función de la aplicación y del número de husillos. Implícitamente, el cofre está en modo asincrónico.



Las dos herramientas conectadas con el cofre constituyen dos máquinas independientes. Los ciclos y los mandos son independientes. Si posteriormente cambian el modo de funcionamiento, se borrarán todos los datos que ya estén guardados.

Figura 44. Modo de funcionamiento asíncrono.

Modo de funcionamiento síncrono



La o las herramientas conectadas con el cofre constituyen una sola y misma máquina. Utilizan uno o varios ciclos comunes pero su parametrización puede ser distinto. Los mandos son comunes para las 2 herramientas.

Figura 45. Modo de funcionamiento síncrono

3.6.3. Tipos de apriete y desapriete

- *Tipos de apriete:* Par, Par+Angulo, Angulo+Par, Par+Angulo+Pendiente, Angulo+Par+Pendiente, Calado al par, Par de rozamiento, Límite elástico/zona plástica. Opcional, control de la corriente.

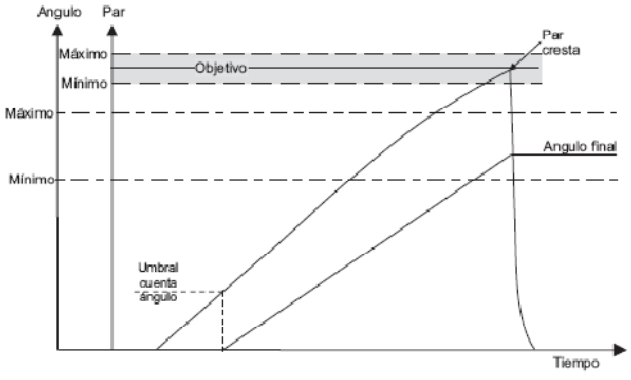


Figura 46. Ejemplo de curva de apriete

CURVAS DE APRIETE

Se puede elegir la curva que se quiere visualizar en función del tipo de curva que se quiere para el proceso de atornillado.
Se puede elegir el modo de representación de la curva:
• par en función del tiempo.
• par+ángulo en función del tiempo.
• par+ángulo+pendiente en función del tiempo.
• par en función del ángulo.

- *Tipos de desapriete:* Par, Par+Angulo, Angulo+Par.

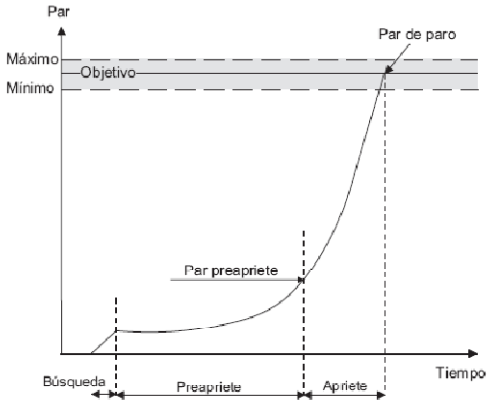


Figura 47. Curva de apriete al par

3.6.3.1. Apriete al par

El valor registrado es: el par cresta
Parada del husillo:

- si par ≥ par de parada

Informe Bueno

- SI par mínimo ≤ par cresta ≤ par máximo

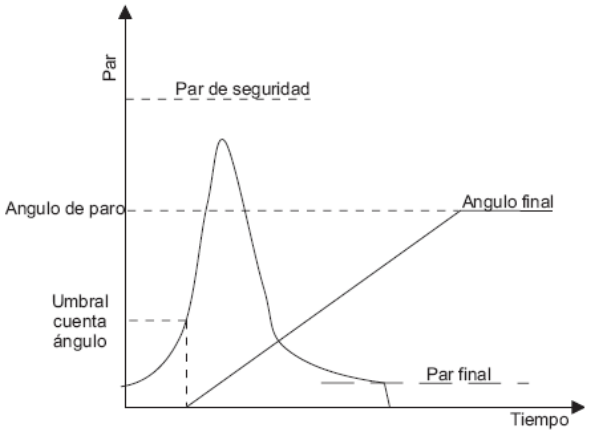
Opcional: Control corriente
- SI par mínimo ≤ par cresta ≤ par máximo
- Y corriente mínima ≤ corriente final ≤ corriente máxima.

Ajuste de la banda de paso:

Para conseguir un ensamblaje correcto, hay que aplicar un par conocido, pero al mismo tiempo dicho par ha de convertirse en esfuerzo en el tornillo. Sin embargo, al final de un apriete, en un ensamblaje muy rígido, si se aplica un par importante durante un tiempo muy corto (choque), el par medido puede ser muy elevado sin que el esfuerzo aumente sensiblemente en el tornillo. En dicho caso, la banda de paso no está en adecuación con el ensamblaje y habrá que adaptarla mediante aproximaciones sucesivas. En la mayor parte de los ensamblajes, la banda de paso es de 128 Hz.

El funcionamiento de desapriete es a par máximo.

3.6.3.2. Desapriete al par + ángulo



Además del control de desapriete del tornillo, el sistema controla el número de grados realizados al mismo tiempo que mantiene un par residual en el tornillo.
El control empieza SI par > par de despegue
Los valores que se guarda son los siguientes: el par final y el ángulo final

Figura 48. Curva de apriete al par + ángulo

Parada del husillo:
- SI par ≤ par de parada
- par > par de seguridad
- ángulo > ángulo máximo

Informe Bueno
- Si par < par de seguridad
- Y par mínimo ≤par final ≤ par máximo

- Y ángulo mínimo ≤ ángulo final ≤ ángulo máximo.

Para conseguir esto se introduce en el valor de ángulo 0º.

3.6.4. Elección de ciclo y características

En este proyecto se ha planteado el funcionamiento de apriete al par, ya descrito anteriormente, debido a que la aplicación necesita definir un par de apriete concreto para el buen funcionamiento mecánico de la pieza. Con ello se garantiza que todos los tornillos se aprietan exactamente con el mismo par.

3.6.4.1. Número de ciclos y de fases

El sistema permite llevar a cabo 250 ciclos de apriete de 20 fases cada uno, por husillo. Los ciclos están numerados del 1 al 250. En esta aplicación se definen seis ciclos máximo por tres fases (cada pieza).

3.6.4.2. Capacidad memoria

Entre 2 500 y 8 500 resultados de apriete por máquina en función del número de ciclos y el modo de funcionamiento (asíncrono o síncrono).

3.6.4.3. Número de curvas

Tres curvas por husillo, dos cuyo informe de ciclo es bueno, 1 cuyo informe es malo. En este proyecto se da informe bueno o malo a través del interfaz de Entradas/Salidas incluido en el equipo.

Para información más específica:
(Consultar el manual de operario TWINCVI II).

Esquema de visualización del conjunto del sistema de atornillado

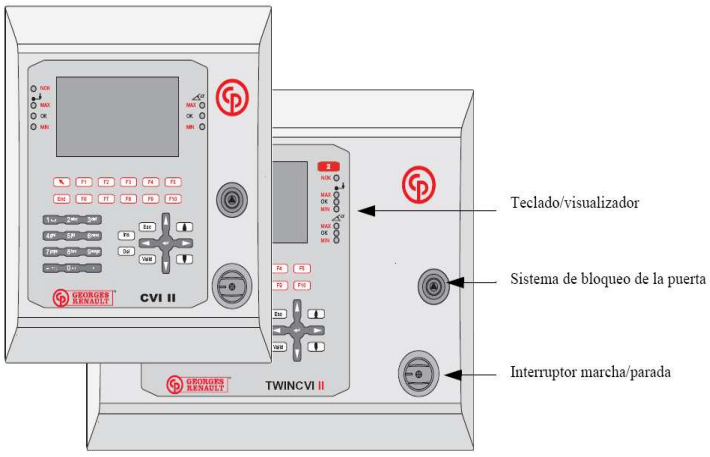


Figura 49. Frontal equipo de atornillado TWINCVI II

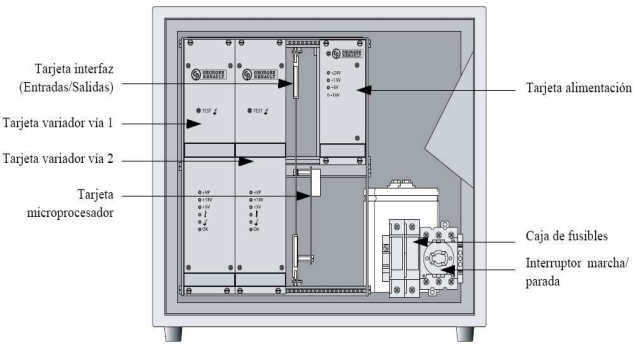


Figura 50. Interior equipo de atornillado TWINCVI II

4. Programación del sistema

4.1. Programación PLC

En este apartado se detallará el programa necesario para la captación de la información de posición del sistema y las órdenes ejecutables hacia el sistema de atornillado automático.

Listado de variables y posición en memoria

Zona de Memoria	Nombre variable	Zona de Memoria	Nombre variable
%M0	uso programa	%MW0	MODO_OPERACION_ENC1
%M1	CANOPEN_OK	%MW1	DIR_CANOPEN_ENC1
%M2	POS_OK_X1	%MW2	DIR_MEM_ENC1
%M3	POS_OK_Y1	%MW3	IND_DIR_ENC1
%M4	POS_OK_X2	%MW4	VALOR_ENC1
%M5	POS_OK_Y2	%MW5	uso programa
%M6	POS_OK_X3	%MW6	uso programa
%M7	POS_OK_Y3	%MW7	uso programa
%M8	POS_OK_X4	%MW8	uso programa
%M9	POS_OK_Y4	%MW9	uso programa
%M10	POS_OK_X5	%MW10	uso programa
%M11	POS_OK_Y5	%MW11	uso programa
%M12	POS_OK_X6	%MW12	MAN_AUT
%M13	POS_OK_Y6	%MW13	INDICADOR_SEQ
%M14	FIN_PLANO	%MW14	MAX_TORNILLOS
%M15	PLANO1_PLC	%MW15	NUM_PLANO
%M16	PLANO2_PLC	%MW16	PLANO_MAX
%M17	PLANO3_PLC	%MW50	CICLO_ATORNILLADOR
%M18	PLANO4_PLC	%MW150	CICLO1_PLC
%M19	PLANO5_PLC	%MW151	CICLO2_PLC
%M20	PLANO6_PLC	%MW152	CICLO3_PLC
%M21	ALCANZADO_PLANOMAX	%MW153	CICLO4_PLC
%M25	FIN_PROCESO	%MW154	CICLO5_PLC
%M30	VALIDACION_MAX_ALC	%MW155	CICLO6_PLC
%M31	POSICION_MAX_ALC_VAL_DE	%MW200	CONTADOR_DEFECTOS
%M32	RESET_CONTADOR	%MW201	NUMERO_VALIDACIONES
%M33	RESET_CONTADOR_HMI	%MW202	uso programa
%M60	POS_ALCANZADA	%MW500	INICIALIZACION_CANOPEN
%M61	MODO_MANUAL	%MW501	INICIALIZACION_CANOPEN
%M65	VALIDACION_DEFECTO		
%M66	PIEZA_NOK		
%M67	TORNILLO_NOK		
%M70	CAMBIO_SEQ		

Zona de Memoria	Nombre variable	Zona de Memoria	Nombre variable
%MD20	PLC_X1	%MD96	PLC_Y1_MENOS_OFFSET_X
%MD22	PLC_Y1	%MD98	PLC_X2_MAS_OFFSET_X
%MD24	PLC_X2	%MD100	PLC_X2_MENOS_OFFSET_X
%MD26	PLC_Y2	%MD102	PLC_Y2_MAS_OFFSET_Y
%MD28	PLC_X3	%MD104	LC_Y2_MENOS_OFFSET_Y
%MD30	PLC_Y3	%MD106	PLC_X3_MAS_OFFSET_X
%MD32	PLC_X4	%MD108	PLC_X3_MENOS_OFFSET_X
%MD34	PLC_Y4	%MD110	PLC_Y3_MAS_OFFSET_Y
%MD36	PLC_X5	%MD112	PLC_Y3_MENOS_OFFSET_Y
%MD38	PLC_Y5	%MD114	PLC_X4_MAS_OFFSET_X
%MD40	PLC_X6	%MD116	PLC_X4_MENOS_OFFSET_X
%MD42	PLC_Y6	%MD118	PLC_Y4_MAS_OFFSET_Y
%MD44	OFFSET_X	%MD120	PLC_Y4_MENOS_OFFSET_Y
%MD46	OFFSET_Y	%MD122	PLC_X5_MAS_OFFSET_X
%MD80	uso programa	%MD124	PLC_X5_MENOS_OFFSET_X
%MD82	POSICION_ENC1	%MD126	PLC_Y5_MAS_OFFSET_Y
%MD84	uso programa	%MD128	PLC_Y5_MENOS_OFFSET_Y
%MD86	POSICION_ENC2	%MD130	PLC_X6_MAS_OFFSET_X
%MD90	PLC_X1_MAS_OFFSET_X	%MD132	PLC_X6_MENOS_OFFSET_X
%MD92	PLC_X1_MENOS_OFFSET_X	%MD134	PLC_Y6_MAS_OFFSET_Y
%MD94	PLC_Y1_MAS_OFFSET_Y	%MD136	PLC_Y6_MENOS_OFFSET_Y

Tabla 10. Listado de entradas/salidas y variables PLC

%M: Zona de memoria tamaño 1 bit
%MW: Zona de memoria tamaño16 bit
%MD: Zona de memoria tamaño 32bit

4.1.1. Programa de aplicación de PLC en Twidosuite

Siguiendo la información del *Manual de programación de Twidosuite*, se puede ampliar la información acerca del manejo de este programa.

En este apartado se describe funcionalmente el programa del PLC. Seguidamente se describen las secciones de programa:

4.1.1.1. Sección 1 y 2: Encoder Can open

En esta sección de programa se inicializa la comunicación con los dos encoder en comunicación Can open.

4.1.1.2. Sección 3: Elección de ciclo PLC-Atornillador

- **Red 0 a Red 4**

Debido a que la elección del ciclo del atornillador automático se realiza mediante señales de entrada-salida, en esta sección se escoge una dirección de memoria (en este caso la %MW50) y se asigna el valor a las salidas físicas del PLC, que transmitirán automáticamente al atornillador automático al ser una comunicación cableada.

- **Red 5 a Red 10**

En este apartado se asignan los valores de ciclo de PLC según el valor de índice de secuencia indicado desde el HMI

4.1.1.3. Sección 4: Paso de datos de Encoder a variables

En esta sección se trasladan los datos de posición de los encoder a zonas de memoria para poder tratarlas posteriormente

4.1.1.4. Sección 5: Validación de tornillos/pieza

- **Red 0 a Red 1**

En esta parte se indica si el proceso de atornillado no se ha realizado correctamente y, por tanto, si la pieza no es correcta.

- **Red 2 a Red 7**

Aquí se programa si se efectúa un movimiento manual (acciona el atornillador sin condiciones) o se inicia el proceso automático.

4.1.1.5. Sección 6: Proceso automático

- **Red 0 a Red 23**

En este apartado se define cual es la posición de los tornillos. Se añade el concepto de Offset (holgura), ya que, debido a la elevada precisión de lo encoder, sería casi imposible repetir exactamente la misma posición, por lo que se le añade un margen de maniobra.

- **Red 24 a Red 35**

Se realiza la comparación de posición actual con la posición memorizada anteriormente y se indica si es correcta.

- **Red 36**

Se define la posición actual como correcta.

- **Red 37 a Red 42**

Se repite el proceso anterior, aumentando progresivamente de secuencia, hasta terminar totalmente con los tornillos memorizados.

- **Red 43 a Red 48**

Se indica el número de plano de la pieza en el que se encuentra el proceso.

- **Red 49 a Red 61**

Se terminan los detalles de programa, como la indicación de fin de trabajo, contador de fallos y borrado de los mismos, etc…

Con esta aplicación se consigue recoger los datos de ambos encoder para determinar su posición y avisar al sistema de atornillado automático que la herramienta se encuentra en posición correcta. Así mismo, el PLC indica a través de entradas/salidas la acción correspondiente a efectuar.

La aplicación desarrollada en Twidosuite se encuentra en el Anexo I de este proyecto.

4.2. Programa de aplicación de HMI en Vijeo Designer

Debido a que se trata de un interfaz gráfico, esta aplicación y su programación se puede encontrar en el Anexo II de este proyecto.
Con esta aplicación se consigue introducir los datos que se quiere en el sistema de atornillado, consiguiendo así el mayor grado de flexibilidad para el proceso.

5. RESULTADOS OBTENIDOS

Para controlar el proceso de atornillado se ha instalado una unidad de atornillado TWINCVI II, parametrizado con apriete al par y controlado por interfaz de Entradas/Salidas. A su vez, se ha instalado un PLC Twido de la marca Schneider, para ejercer las funciones de control hacia el atornillador eléctrico y obtener la información del mismo. Con esto, a través de una zona de memoria del PLC, se puede actuar sobre el atornillador y modificar cualquiera de sus funciones de forma remota. A su vez, se han instalado dos encoder de comunicación CanOpen de la gama Osicoder de Schneider Electric,para la transmisión de la posición de la herramienta de atornillado, la cual es recogida y posteriormente tratada por el PLC. Esto permite conocer la posición del atornillador en los ejes X e Y, consiguiendo delimitar las actuaciones del operario. A su vez, se han implementado funciones de Offset a la ventana de actuación de apriete del tornillo.

Se ha instalado un interfaz táctil XBTGT de la marca Schneder Electric en el PLC, a través de comunicación Modbus RTU, pudiendo visualizar los datos recogidos en el PLC, y consiguiendo escoger los parámetros de posición y funcionamiento del atornillador eléctrico. El terminal también se ha utilizado como medio de parametrización de la aplicación, reduciendo a un único punto este cometido. En este terminal se puede escoger el orden de operación de atornillado, así como el par de apriete a efectuar.

Con esto, se ha conseguido controlar el proceso de atornillado, tanto la posición de la operación como el orden atornillado. Se ha conseguido parametrizar variables como el par de apriete o la posición de los tornillos a atornillar. El operario puede decidir cuál va a ser el orden de la operación, el par de apriete y puede supervisar que toda la operación se está desarrollando de forma correcta.

6. CONCLUSIONES

Habiendo integrado todas las partes del sistema se concluye que es posible la integración de elementos para mejorar el proceso de atornillado automático y cumplir con las necesidades que exige el mercado actualmente.

En el presente proyecto se ha integrado en el proceso de atornillado un sistema de control y supervisión que permite a través de controles estándar de mercado configurar una nueva máquina industrial. Se ha integrando un PLC para el control global de la máquina, dos encoder de comunicación industrial para transmitir la posición del atornillador y pantalla HMI para el control y supervisión del proceso, y un atornillador automático para desarrollar la parte del atornillado mecánico del proceso. En el proyecto se realiza un ejemplo práctico donde se desarrolla un sistema de comunicación entre los distintos elementos que permitirá realizar todas las especificaciones del proyecto.

En el proyecto se ha realizado la configuración de una red de comunicación para desarrollar la completa integración de los equipos en un sistema.

Con este tipo de sistemas se puede concluir que se puede añadir un carácter flexible a cualquier tipo de máquina.

7. DESARROLLOS FUTUROS

El ejemplo práctico de este proyecto se limita a una sola máquina para una pieza definida con tres planos de atornillado. Se puede desarrollar de una forma relativamente sencilla una ampliación a seis planos de atornillado, lo que permitiría una flexibilización para cualquier pieza.

Normalmente en los procesos industriales se pueden encontrar distintos procesos, los cuales están ligados unos con otros. Debido al diseño de este sistema, se podrían transmitir datos de unos procesos a otros para el mejor funcionamiento de los mismos, o para obtener una trazabilidad de procesos.

En esta línea de mejora, se podrían escalar todos los datos relacionados con los distintos procesos (atonillado, mecanizado, extrusión, montaje,etc..) a un sistema de gestión superior como un SCADA (programa de supervisión y control a través de un PC), así como implementar comunicaciones de mayor rendimiento como Ethernet.

8. Análisis de costes

Se realiza una valoración de los recursos materiales y técnicos que se van a requerir en este proyecto. El cálculo de tiempo de amortización de la operación se calcula para un volumen de 5 máquinas. El coste de operario es de 8€/hora.

8.1. Los recursos materiales

En este análisis se estima necesario:

Descripción	Cantidad	Precio unitario	Precio total
PLC Twido TWDLCAA24DRF	**1**	**450 €**	**450 €**
HMI Magelis XBTGT2220	**1**	**650 €**	**650 €**
Atornillador TWIN CVI II	**1**	**10.300 €**	**10.300 €**
Encoder Canopen XCC3510PS84CB	**2**	**1.300 €**	**2.600 €**
Cable Modbus 1m. VW3A8306R10	**1**	**19 €**	**19 €**
Cable Canopen FTXCN3210	**2**	**38 €**	**76 €**
Software Twidosuite	**1**	**0 €**	**0 €**
Software Vijeo Designer	**1**	**620 €**	**620 €**
Hora de programación	**350**	**35 €**	**12.250 €**
Puesta en marcha	**1**	**200 €**	**200 €**

8.2. Cálculo de los costos

8.2.1. Situación inicial

La empresa trabaja tres turnos a ocho horas por turno. La cadencia de producción inicial es de 45 piezas/hora. La tasa de piezas válidas (superan calidad) es del 85%. Inicialmente se necesita calcular el precio unitario por pieza, por lo que, teniendo en cuenta los gastos mostrados en la siguiente tabla, se calcula el total de gastos diarios asociados a la producción:

Definición	*Coste*	*Total*
Coste diario trabajador	(10€*16horas)+(15€*8horas)	280€
Energía	((0,1051€/Kwh*0,4Kw)*24h)+1€ consumo otros elementos	2€
Materia prima	43€ diarios	43€
Otros gastos	32€ diario	32€
Total		**358€**

8.2.2. Cálculo del coste por pieza

La cadencia es de 45 piezas/hora, por lo que el número de piezas es

$$Nº\ Piezas = \frac{(\frac{45 piezas}{hora} \times 24 horas)}{5\ máquinas} \times 0{,}85 = 184\ piezas\ por\ máquina\ y\ día$$

El coste final por pieza sería:

$$\boldsymbol{Coste = \frac{358€}{184\ piezas} = 1{,}95\ €\ por\ pieza}$$

Al instalar la máquina de atornillado, se pasa a una tasa de piezas válidas del 98%, por lo que el coste final con automatización, en cuanto a producción, gracias a la mejora, la cadencia de piezas a la hora pasa a ser de 60 piezas/hora, con lo que el coste de la pieza pasa a ser:

$$Nº\ Piezas = \frac{(\frac{60 piezas}{hora} \times 24 horas)}{5\ máquinas} \times 0{,}98 = 282\ piezas\ por\ máquina\ y\ día$$

$$\boldsymbol{Coste = \frac{358€}{282\ piezas} = 1{,}27\ €\ por\ pieza}$$

Con el consecuente ahorro de materias primas.

8.2.3. Cálculo del beneficio

La inversión inicial por máquina sería aproximadamente de 35315€, con un beneficio de venta para el fabricante de un 30% incluido.

El beneficio anual por máquina para el Usuario Final sería de:

$$Bf = \left(\left(\frac{1{,}95€}{pieza} - \frac{1{,}27€}{pieza}\right) \times \left(\frac{282 piezas}{dia} - \frac{184 piezas}{dia}\right)\right) \times 365 dias = 24502{,}53€$$

Aplicando un factor corrector del 10%, debido a máquina parada, pausa de producción, mantenimiento, etc… El beneficio anual final para el Usuario será de:

$$\boldsymbol{Bf_c = 22052{,}28€}$$

8.3. Retorno de la inversión

Para calcular el período de amortización se seguirá la siguiente técnica:

Año	Coste	Beneficio	Beneficio neto
0	**C0**	**0**	**0**
1	**C1**	**B1**	**B1 – C1**
2	**C2**	**B2**	**B2 – C2**
…	**…**	**…**	**…**
N	**Cn**	**Bn**	**Bn-Cn**

El año de recuperación de la inversión se produce cuando Σ Beneficio Neto = C0.

El tiempo para recuperar la inversión de las cinco máquinas:

Año	Coste	Beneficio neto	Diferencial
0	**176.573 €**	**0**	**-176.573 €**
1	**0 €**	**122512,67**	**-54.060 €**
2	**0 €**	**122512,67**	**68.453 €**
3	**0 €**	**122512,67**	**190.966 €**
4	**0 €**	**122512,67**	**313.478 €**

Según este cálculo, el retorno de la inversión se producirá dentro del tercer año desde su realización.

9. Bibliografía

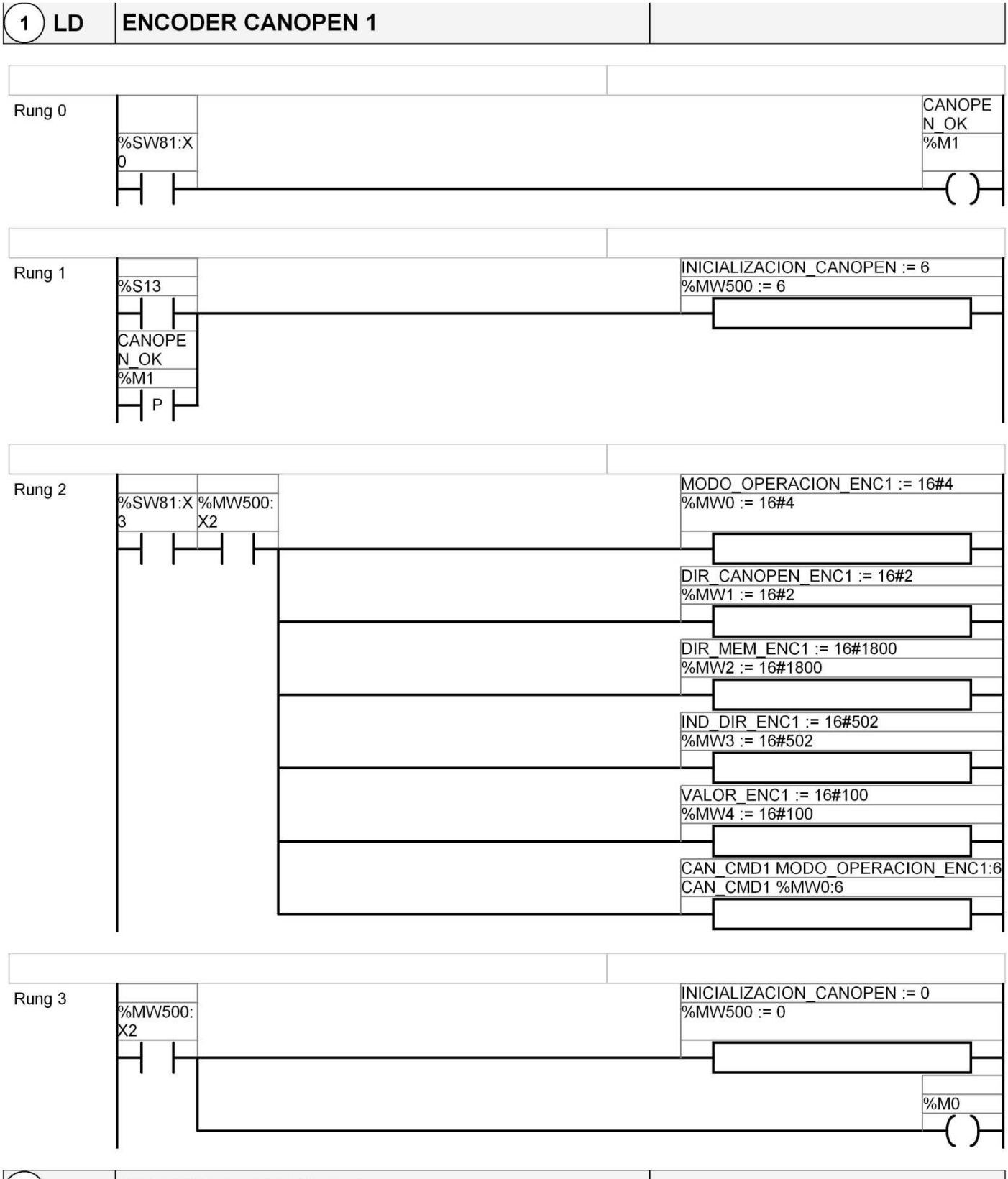
CAN in Automation (CiA). (2010). Obtenido de http://www.can-cia.org
GEORGES RENAULT S.A.S. (2004). Manual de operario TWINCVI II / CVI II version 4.1. *6159932220-02* .
Modbus Organization. (s.f.). Recuperado el 2010, de http://www.modbus.org
Schneider Electric. (2010). Catálogo Osicoder MKTED210041ES.
Schneider Electric. (2008). Catálogo Twido Schneider Electric nº 440038 C08.
Schneider Electric. (s.f.). Manual de curso auto aprendizaje Twidosuite.
Schneider Electric. (s.f.). Manual de curso Vijeo Designer nivel I.
Schneider Electric. (s.f.). Manual de curso Vijeo Designer nivel II.
Schneider Electric. (2008). Manual de programación Twidosuite.
Schneider Electric. (2006). Manual Twido CanOpen instalación nº 35008787 00.
Shoup, T. E., & Spotts, M. F. (1999). *Elementos de máquinas.* Prentice Hall.

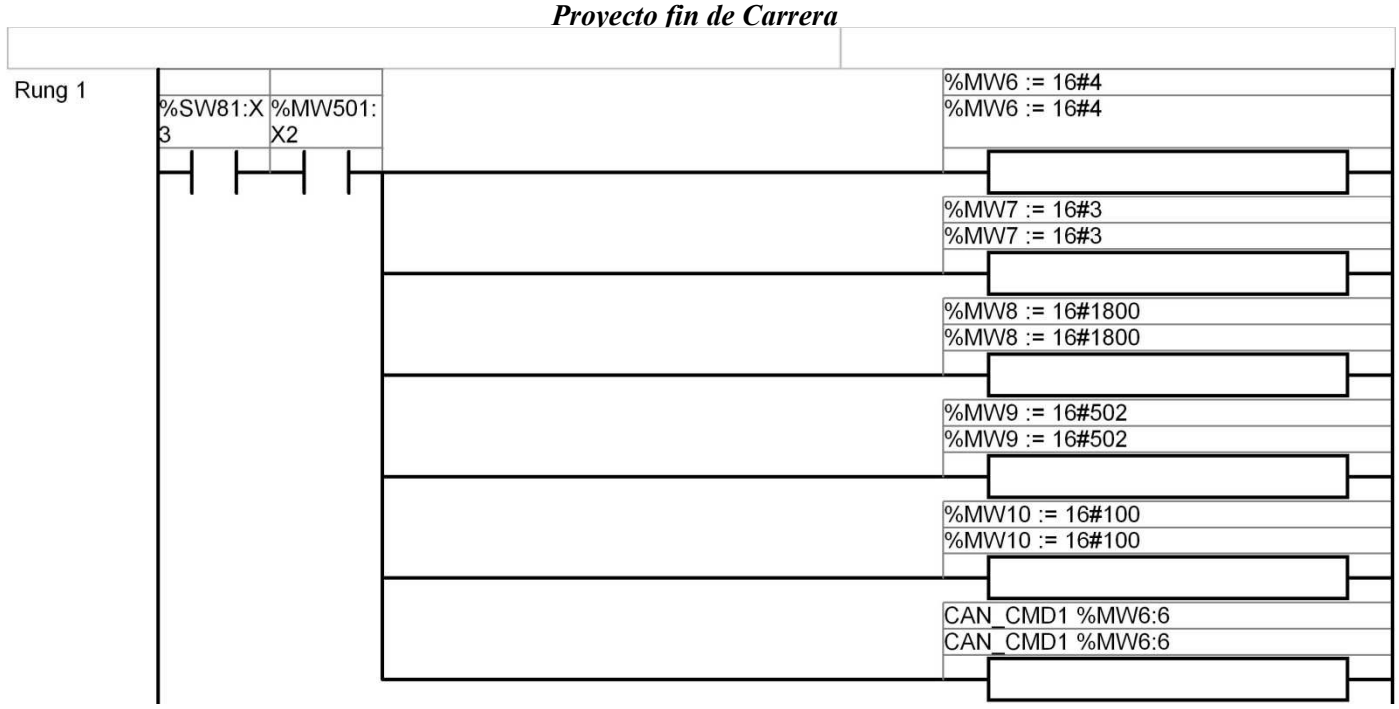
Para información adicional también se estuvo n contacto con la web oficial de Schneider Electric *www.schneiderelectric.es* donde se encontraron datos importantes para la realización de la aplicación y a la hora de la resolución de problemas surgidos durante el desarrollo.

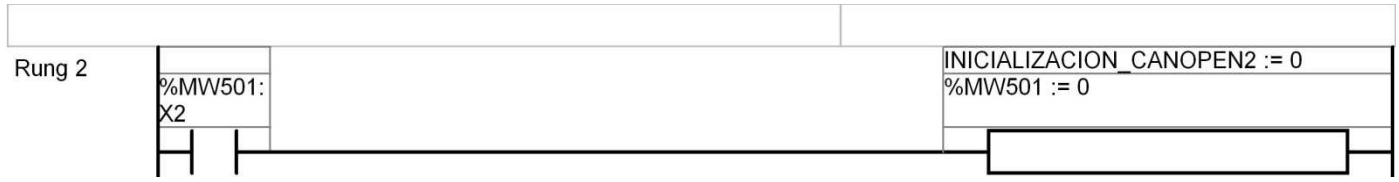
Notas:

Notas:

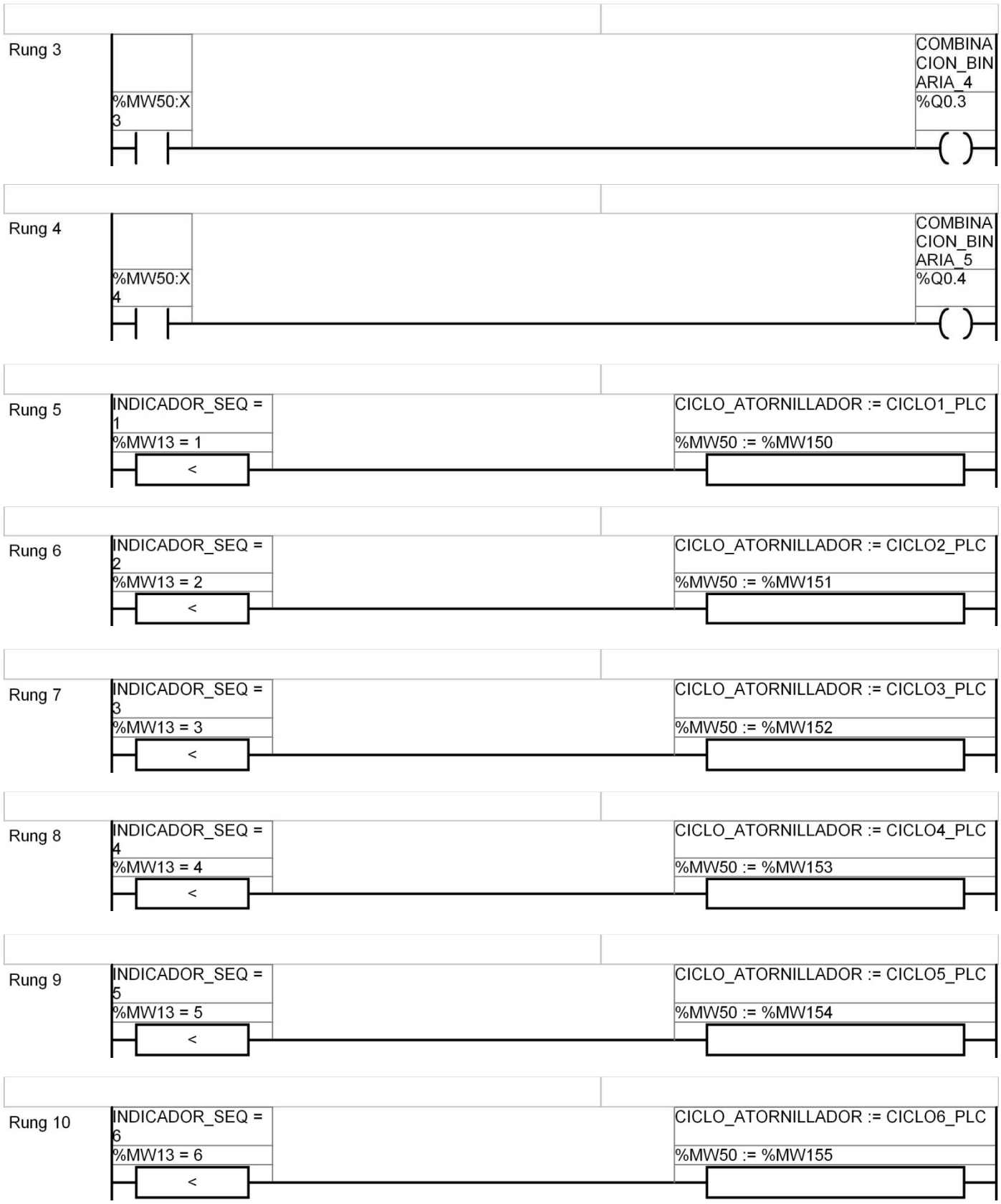
ANEXO I: Programación PLC en Ladder (Twidosuite)


1 LD
ENCODER CANOPEN 1
Rung 0
%SW81:X0
CANOPEN_OK
%M1
Rung 1
%S13
CANOPEN_OK
%M1
P
INICIALIZACION_CANOPEN := 6
%MW500 := 6
Rung 2
%SW81:X3
%MW500:X2
MODO_OPERACION_ENC1 := 16#4
%MW0 := 16#4
DIR_CANOPEN_ENC1 := 16#2
%MW1 := 16#2
DIR_MEM_ENC1 := 16#1800
%MW2 := 16#1800
IND_DIR_ENC1 := 16#502
%MW3 := 16#502
VALOR_ENC1 := 16#100
%MW4 := 16#100
CAN_CMD1 MODO_OPERACION_ENC1:6
CAN_CMD1 %MW0:6
Rung 3
%MW500:X2
INICIALIZACION_CANOPEN := 0
%MW500 := 0
%M0
2 LD
ENCODER CANOPEN2
Rung 0
%M0
N
INICIALIZACION_CANOPEN2 := 6
%MW501 := 6



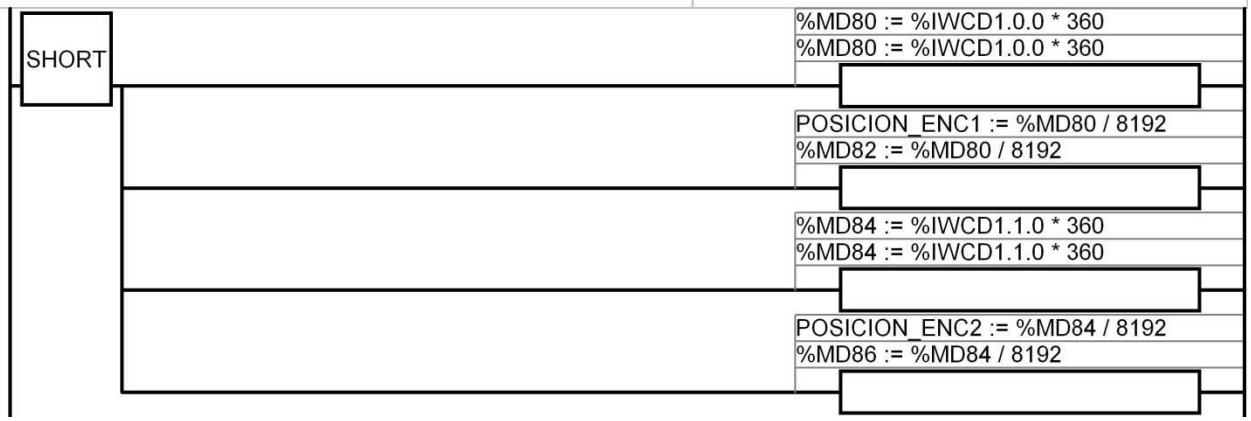


3 LD **ELECCION DEL CICLO DE PLC_ATORNILLADOR**

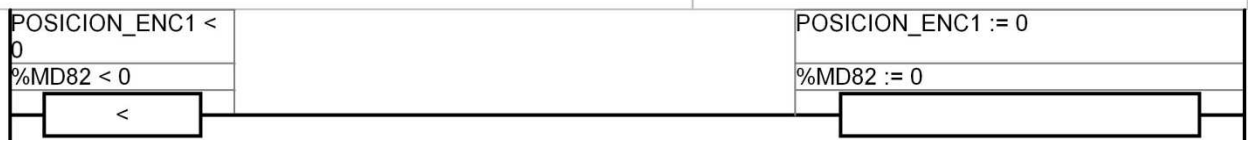


4 LD	PASO DE DATOS ENCODER A VARIABLES	

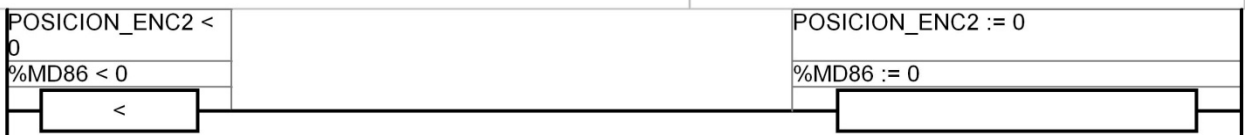
Rung 0



Rung 1

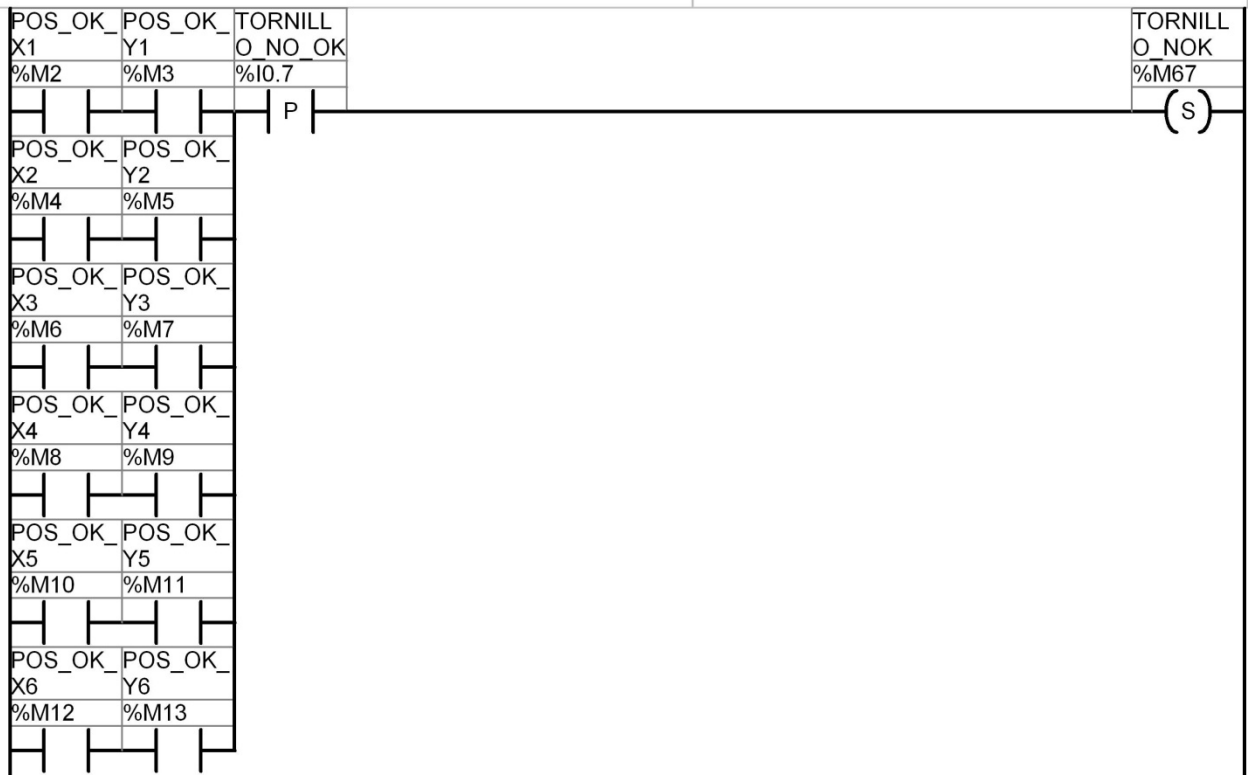


Rung 2

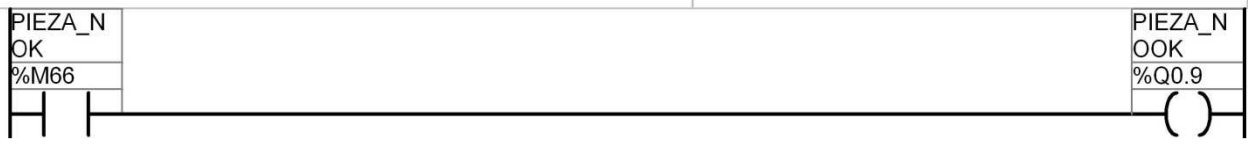


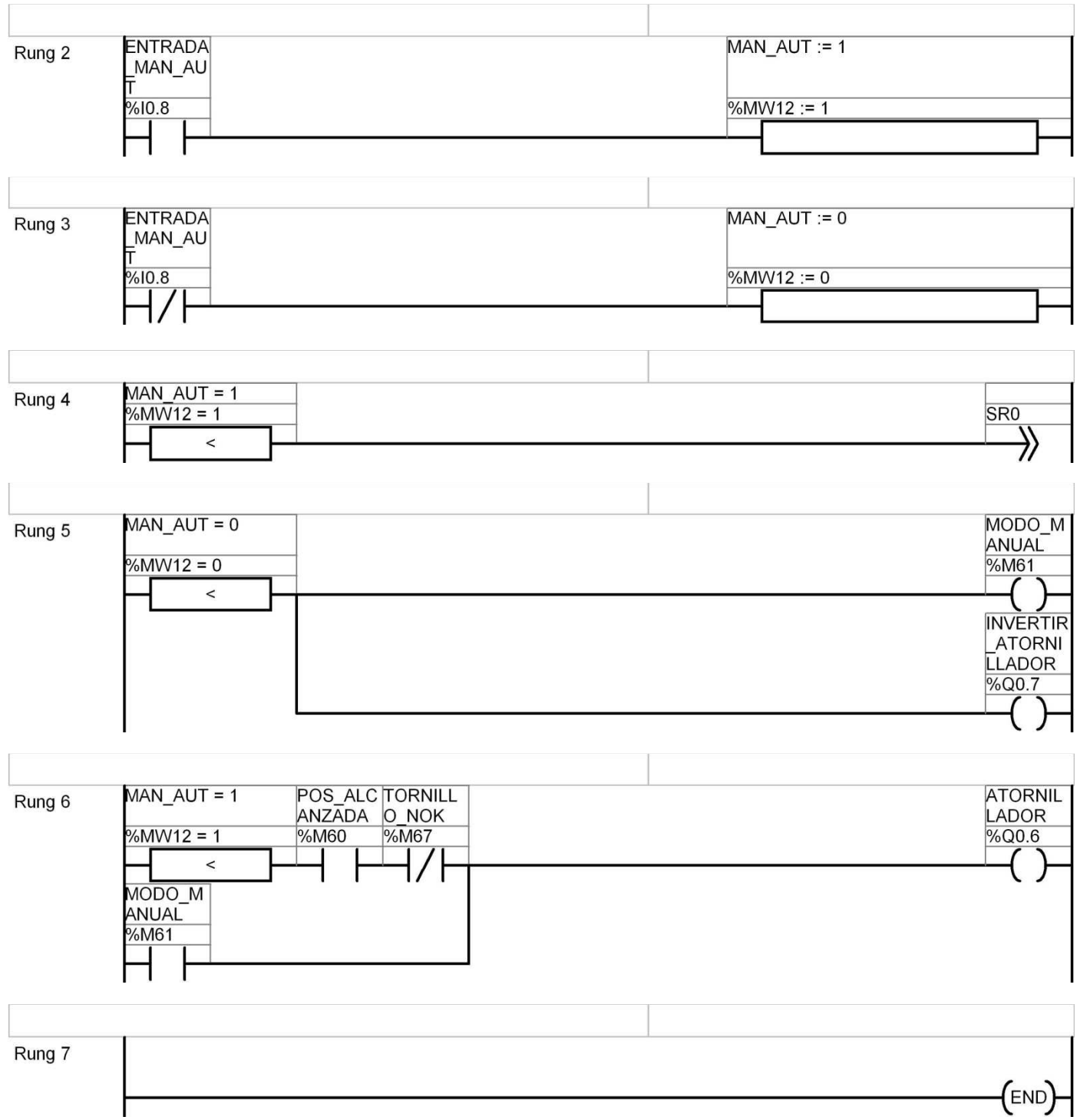
5 LD

Rung 0

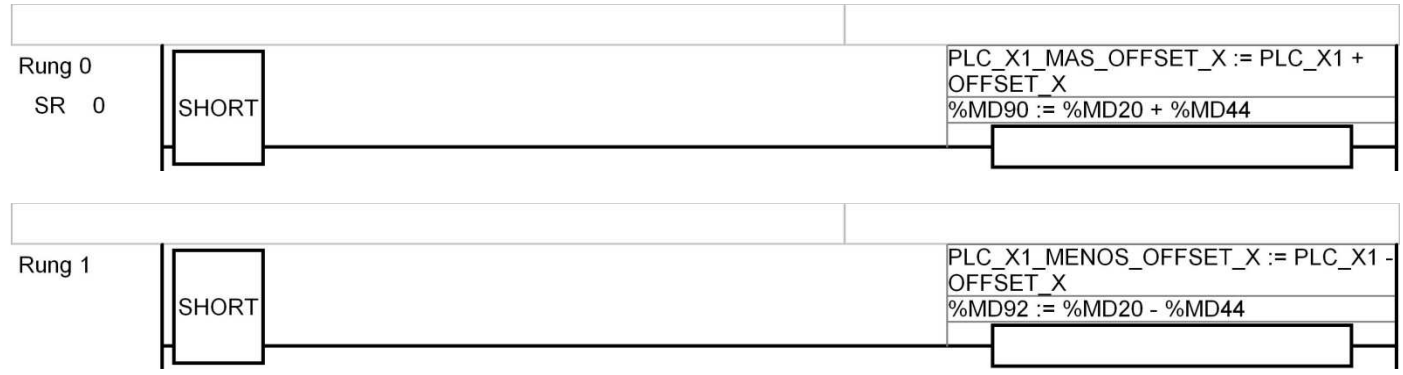


Rung 1




Rung 2
ENTRADA_MAN_AUT
%I0.8
MAN_AUT := 1
%MW12 := 1
Rung 3
ENTRADA_MAN_AUT
%I0.8
MAN_AUT := 0
%MW12 := 0
Rung 4
MAN_AUT = 1
%MW12 = 1
<
SR0
Rung 5
MAN_AUT = 0
%MW12 = 0
<
MODO_MANUAL
%M61
INVERTIR_ATORNILLADOR
%Q0.7
Rung 6
MAN_AUT = 1
%MW12 = 1
<
POS_ALCANZADA
%M60
TORNILLO_NOK
%M67
MODO_MANUAL
%M61
ATORNILLADOR
%Q0.6
Rung 7
END

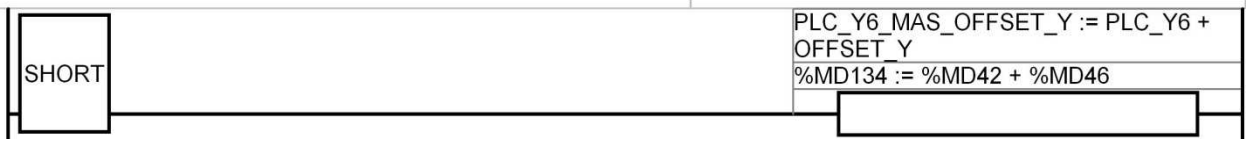
6 LD


Rung 0
SR 0
SHORT
PLC_X1_MAS_OFFSET_X := PLC_X1 + OFFSET_X
%MD90 := %MD20 + %MD44
Rung 1
SHORT
PLC_X1_MENOS_OFFSET_X := PLC_X1 - OFFSET_X
%MD92 := %MD20 - %MD44

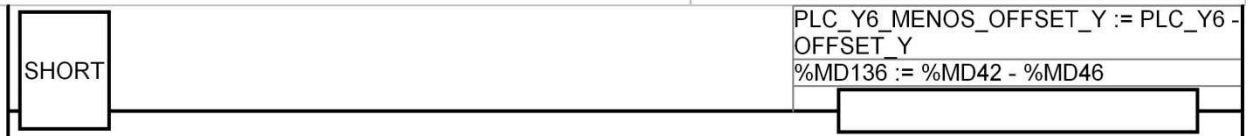

Rung 2
SHORT
PLC_Y1_MAS_OFFSET_Y := PLC_Y1 + OFFSET_Y
%MD94 := %MD22 + %MD46
Rung 3
SHORT
PLC_Y1_MENOS_OFFSET_X := PLC_Y1 - OFFSET_Y
%MD96 := %MD22 - %MD46
Rung 4
SHORT
PLC_X2_MAS_OFFSET_X := PLC_X2 + OFFSET_X
%MD98 := %MD24 + %MD44
Rung 5
SHORT
PLC_X2_MENOS_OFFSET_X := PLC_X2 - OFFSET_X
%MD100 := %MD24 - %MD44
Rung 6
SHORT
PLC_Y2_MAS_OFFSET_Y := PLC_Y2 + OFFSET_Y
%MD102 := %MD26 + %MD46
Rung 7
SHORT
PLC_Y2_MENOS_OFFSET_Y := PLC_Y2 - OFFSET_Y
%MD104 := %MD26 - %MD46
Rung 8
SHORT
PLC_X3_MAS_OFFSET_X := PLC_X3 + OFFSET_X
%MD106 := %MD28 + %MD44
Rung 9
SHORT
PLC_X3_MENOS_OFFSET_X := PLC_X3 - OFFSET_X
%MD108 := %MD28 - %MD44
Rung 10
SHORT
PLC_Y3_MAS_OFFSET_Y := PLC_Y3 + OFFSET_Y
%MD110 := %MD30 + %MD46
Rung 11
SHORT
PLC_Y3_MENOS_OFFSET_Y := PLC_Y3 - OFFSET_Y
%MD112 := %MD30 - %MD46


Rung 12
SHORT
PLC_X4_MAS_OFFSET_X := PLC_X4 + OFFSET_X
%MD114 := %MD32 + %MD44
Rung 13
SHORT
PLC_X4_MENOS_OFFSET_X := PLC_X4 - OFFSET_X
%MD116 := %MD32 - %MD44
Rung 14
SHORT
PLC_Y4_MAS_OFFSET_Y := PLC_Y4 + OFFSET_Y
%MD118 := %MD34 + %MD46
Rung 15
SHORT
PLC_Y4_MENOS_OFFSET_Y := PLC_Y4 - OFFSET_Y
%MD120 := %MD34 - %MD46
Rung 16
SHORT
PLC_X5_MAS_OFFSET_X := PLC_X5 + OFFSET_X
%MD122 := %MD36 + %MD44
Rung 17
SHORT
PLC_X5_MENOS_OFFSET_X := PLC_X5 - OFFSET_X
%MD124 := %MD36 - %MD44
Rung 18
SHORT
PLC_Y5_MAS_OFFSET_Y := PLC_Y5 + OFFSET_Y
%MD126 := %MD38 + %MD46
Rung 19
SHORT
PLC_Y5_MENOS_OFFSET_Y := PLC_Y5 - OFFSET_Y
%MD128 := %MD38 - %MD46
Rung 20
SHORT
PLC_X6_MAS_OFFSET_X := PLC_X6 + OFFSET_X
%MD130 := %MD40 + %MD44
Rung 21
SHORT
PLC_X6_MENOS_OFFSET_X := PLC_X6 - OFFSET_X
%MD132 := %MD40 - %MD44

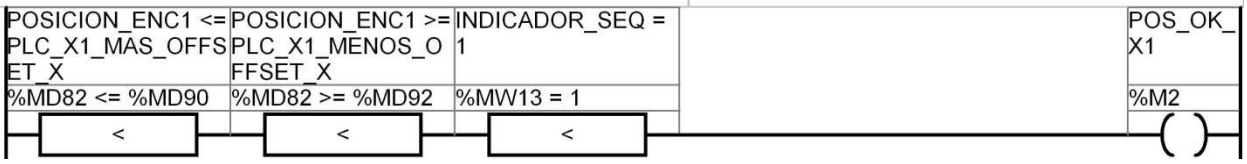
Rung 22



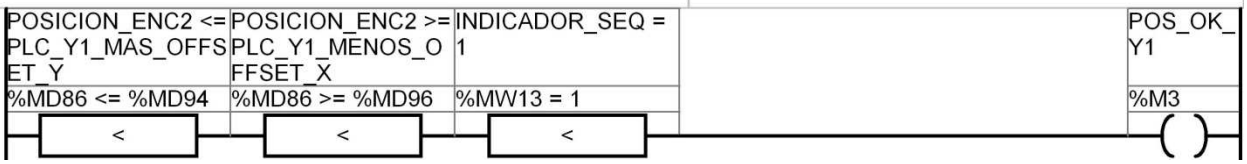
Rung 23



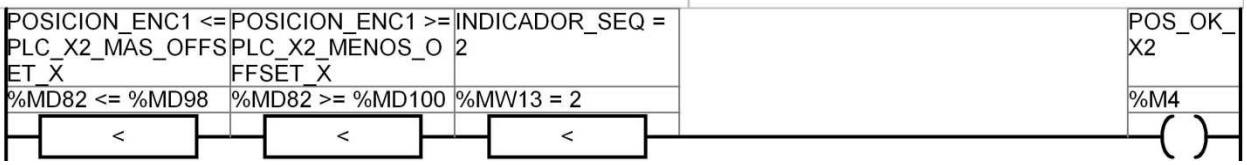
Rung 24



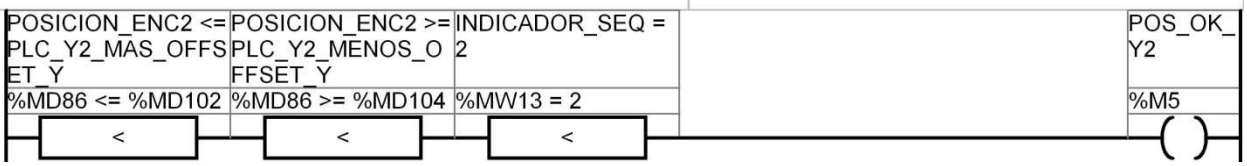
Rung 25



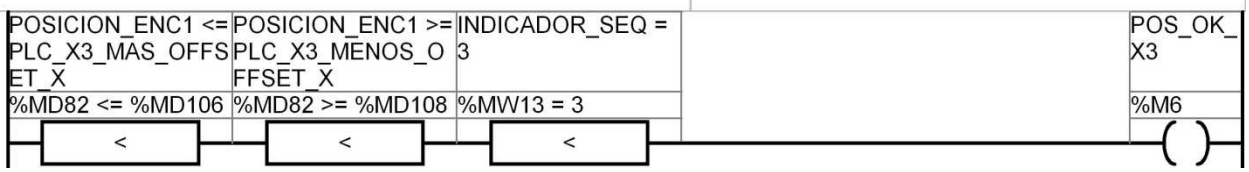
Rung 26



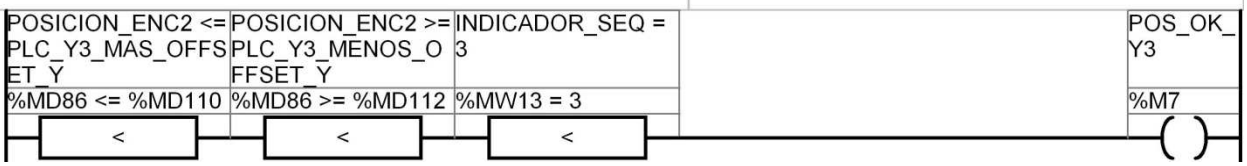
Rung 27



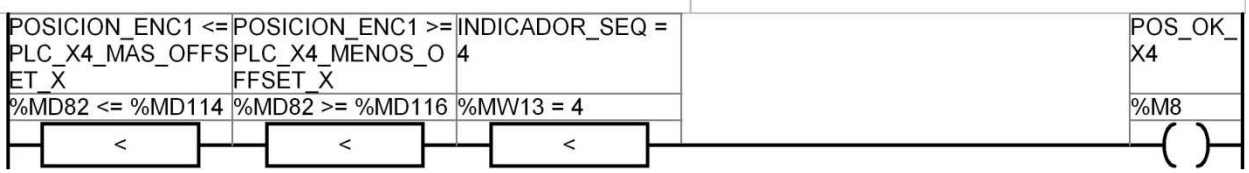
Rung 28

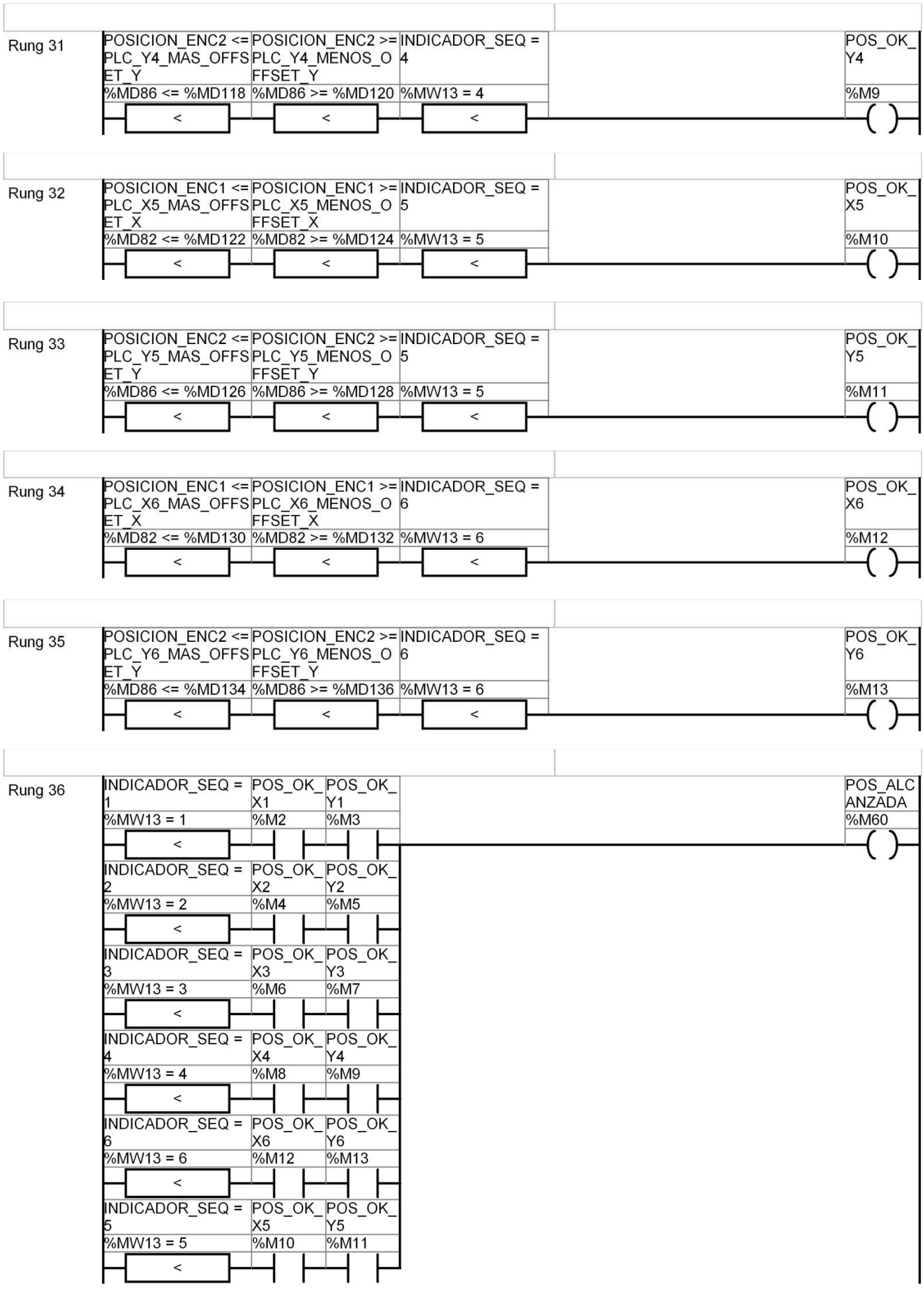


Rung 29

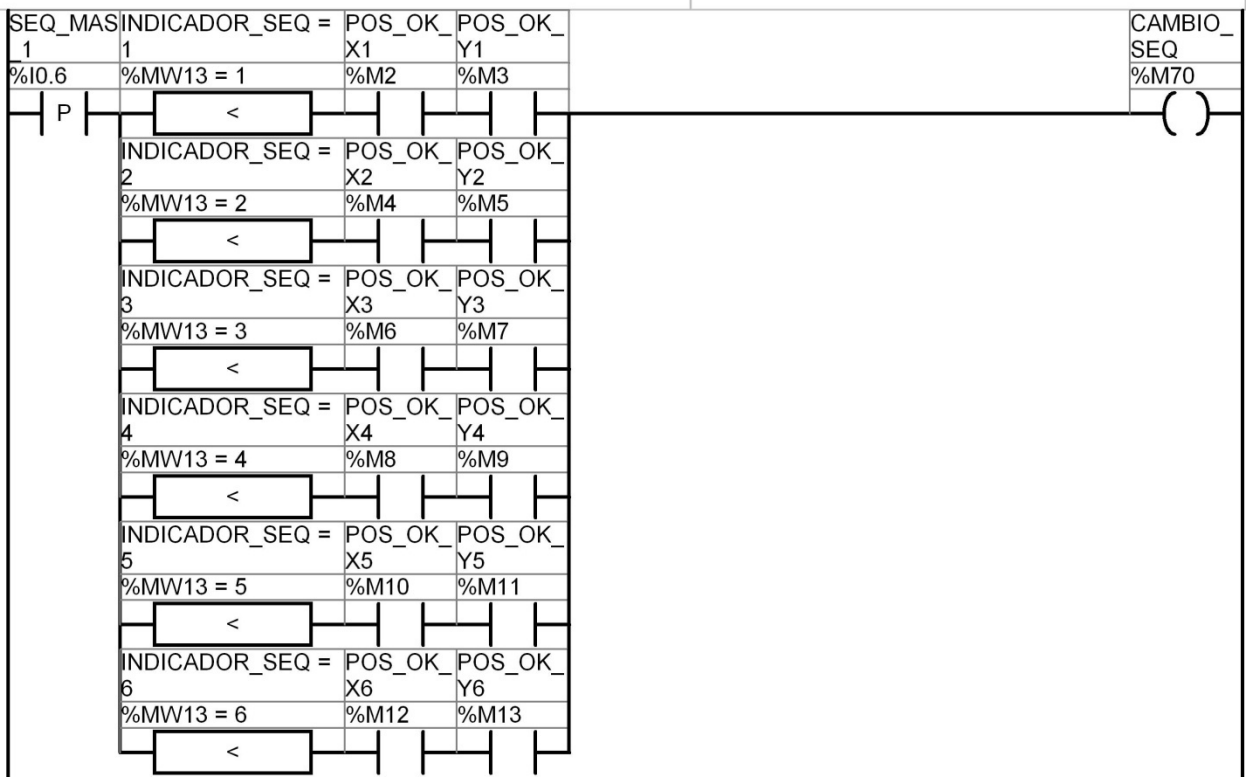


Rung 30

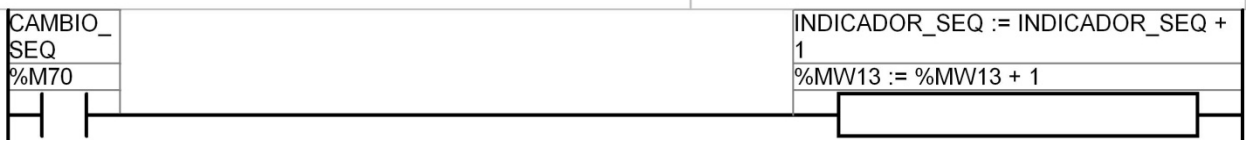



Rung 31
POSICION_ENC2 <= PLC_Y4_MAS_OFFSET_Y
%MD86 <= %MD118
POSICION_ENC2 >= PLC_Y4_MENOS_OFFSET_Y
%MD86 >= %MD120
INDICADOR_SEQ = 4
%MW13 = 4
POS_OK_Y4
%M9
Rung 32
POSICION_ENC1 <= PLC_X5_MAS_OFFSET_X
%MD82 <= %MD122
POSICION_ENC1 >= PLC_X5_MENOS_OFFSET_X
%MD82 >= %MD124
INDICADOR_SEQ = 5
%MW13 = 5
POS_OK_X5
%M10
Rung 33
POSICION_ENC2 <= PLC_Y5_MAS_OFFSET_Y
%MD86 <= %MD126
POSICION_ENC2 >= PLC_Y5_MENOS_OFFSET_Y
%MD86 >= %MD128
INDICADOR_SEQ = 5
%MW13 = 5
POS_OK_Y5
%M11
Rung 34
POSICION_ENC1 <= PLC_X6_MAS_OFFSET_X
%MD82 <= %MD130
POSICION_ENC1 >= PLC_X6_MENOS_OFFSET_X
%MD82 >= %MD132
INDICADOR_SEQ = 6
%MW13 = 6
POS_OK_X6
%M12
Rung 35
POSICION_ENC2 <= PLC_Y6_MAS_OFFSET_Y
%MD86 <= %MD134
POSICION_ENC2 >= PLC_Y6_MENOS_OFFSET_Y
%MD86 >= %MD136
INDICADOR_SEQ = 6
%MW13 = 6
POS_OK_Y6
%M13
Rung 36
INDICADOR_SEQ = 1
%MW13 = 1
POS_OK_X1
%M2
POS_OK_Y1
%M3
INDICADOR_SEQ = 2
%MW13 = 2
POS_OK_X2
%M4
POS_OK_Y2
%M5
INDICADOR_SEQ = 3
%MW13 = 3
POS_OK_X3
%M6
POS_OK_Y3
%M7
INDICADOR_SEQ = 4
%MW13 = 4
POS_OK_X4
%M8
POS_OK_Y4
%M9
INDICADOR_SEQ = 6
%MW13 = 6
POS_OK_X6
%M12
POS_OK_Y6
%M13
INDICADOR_SEQ = 5
%MW13 = 5
POS_OK_X5
%M10
POS_OK_Y5
%M11
POS_ALCANZADA
%M60

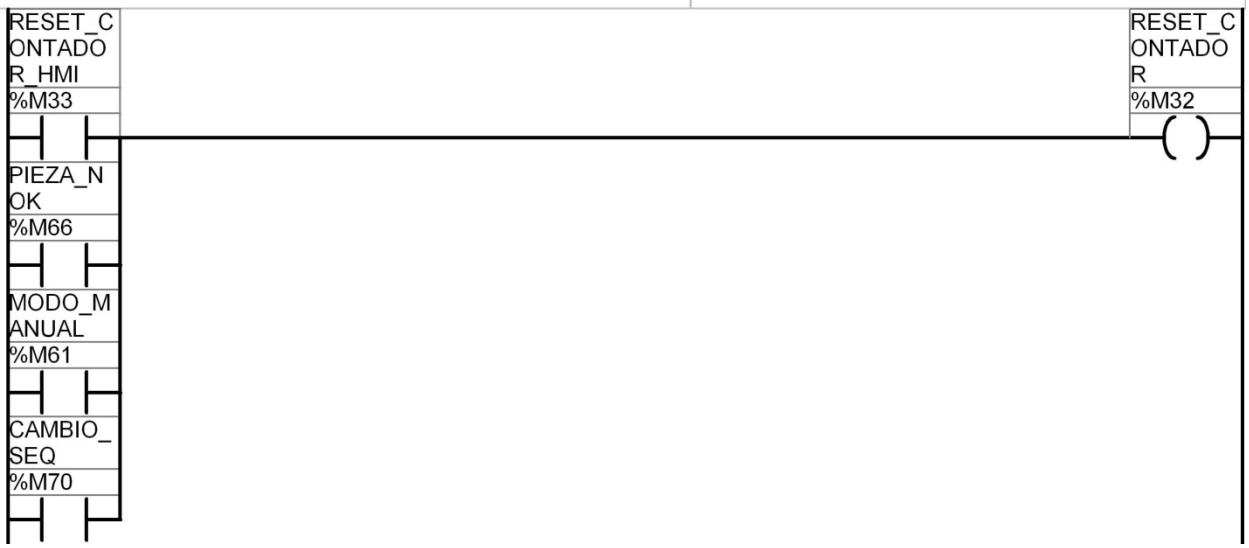
Rung 37



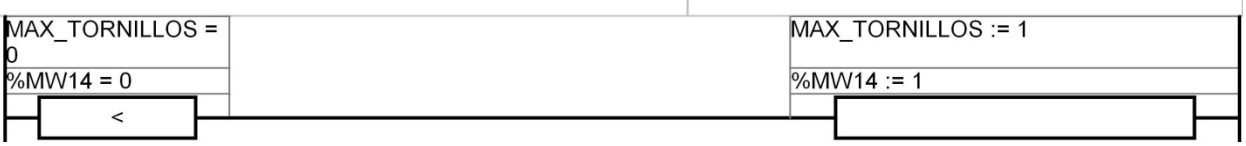
Rung 38



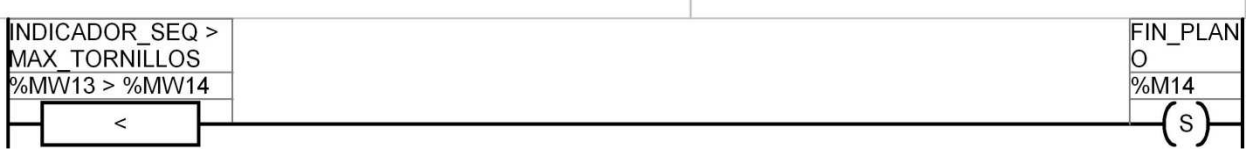
Rung 39

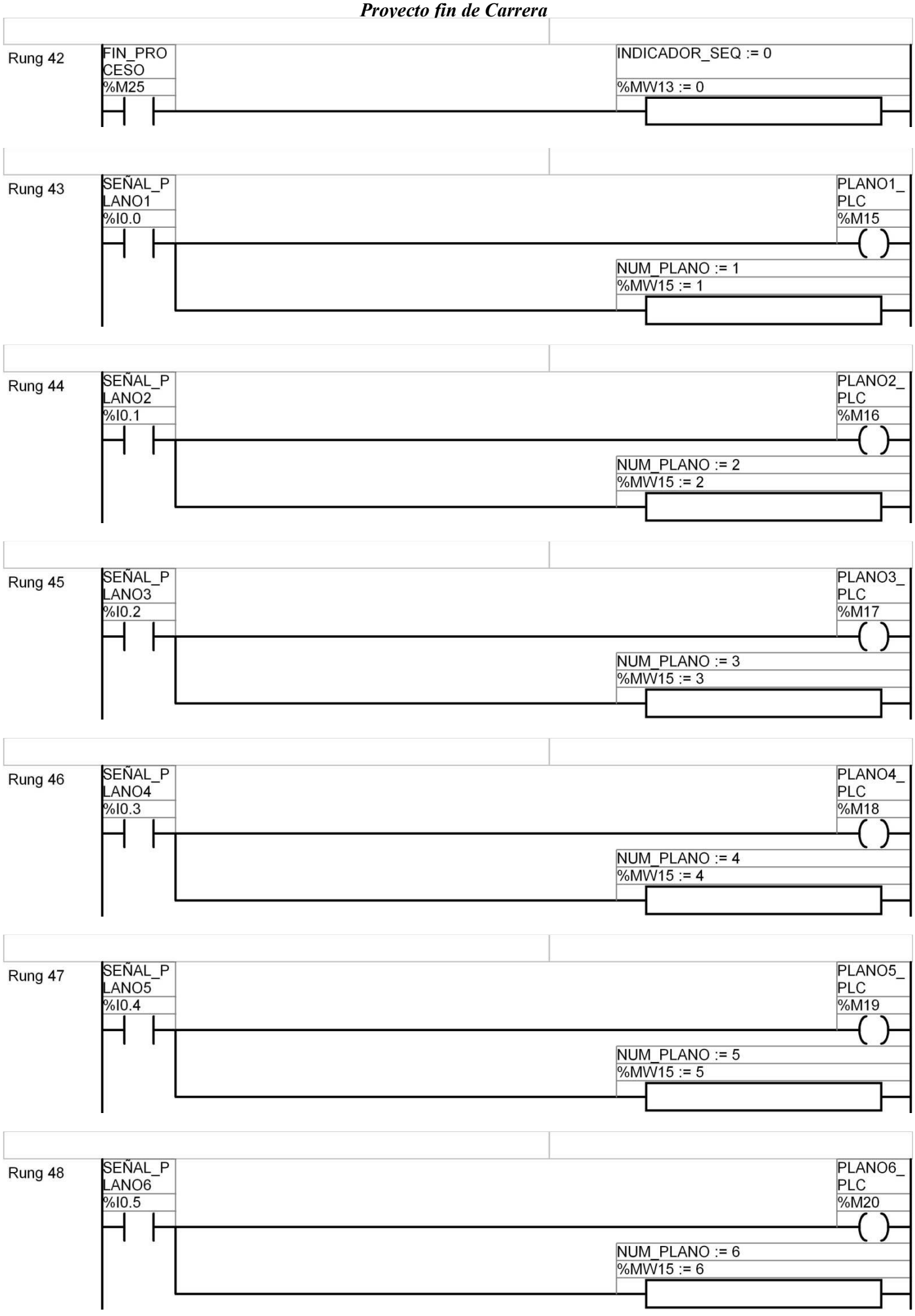


Rung 40

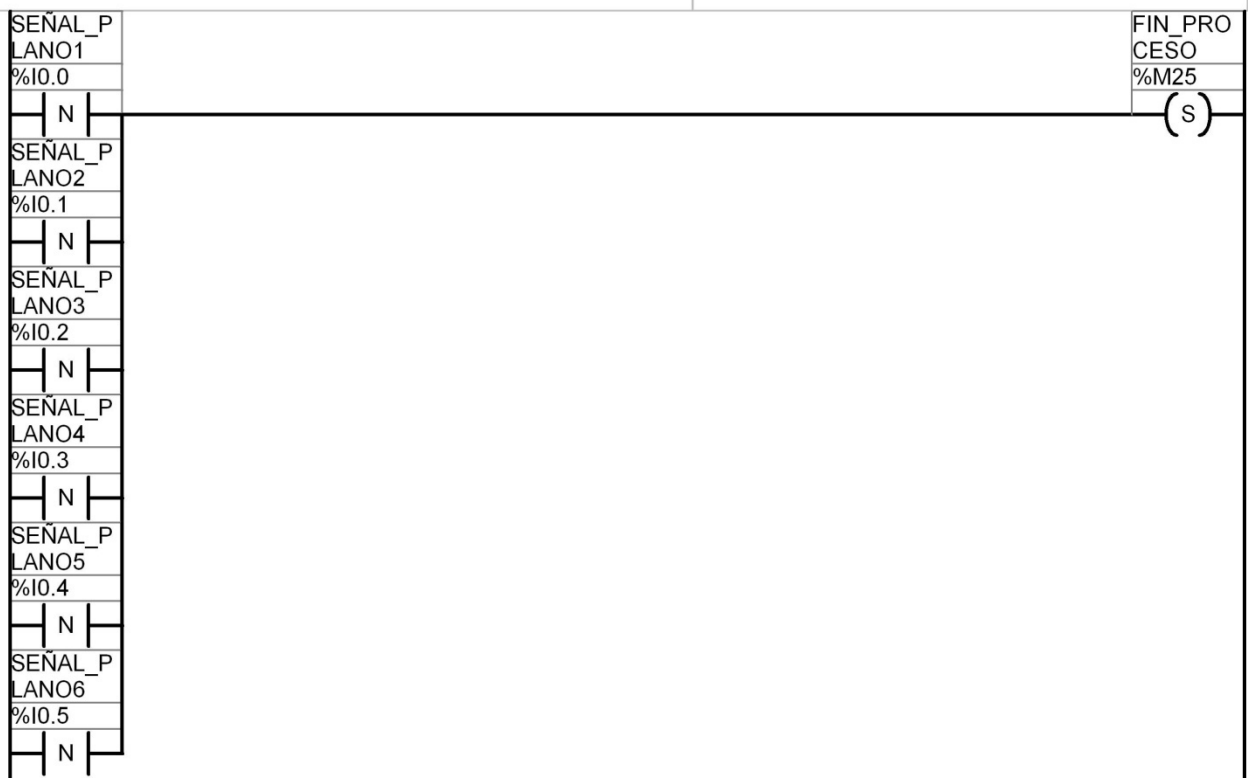


Rung 41

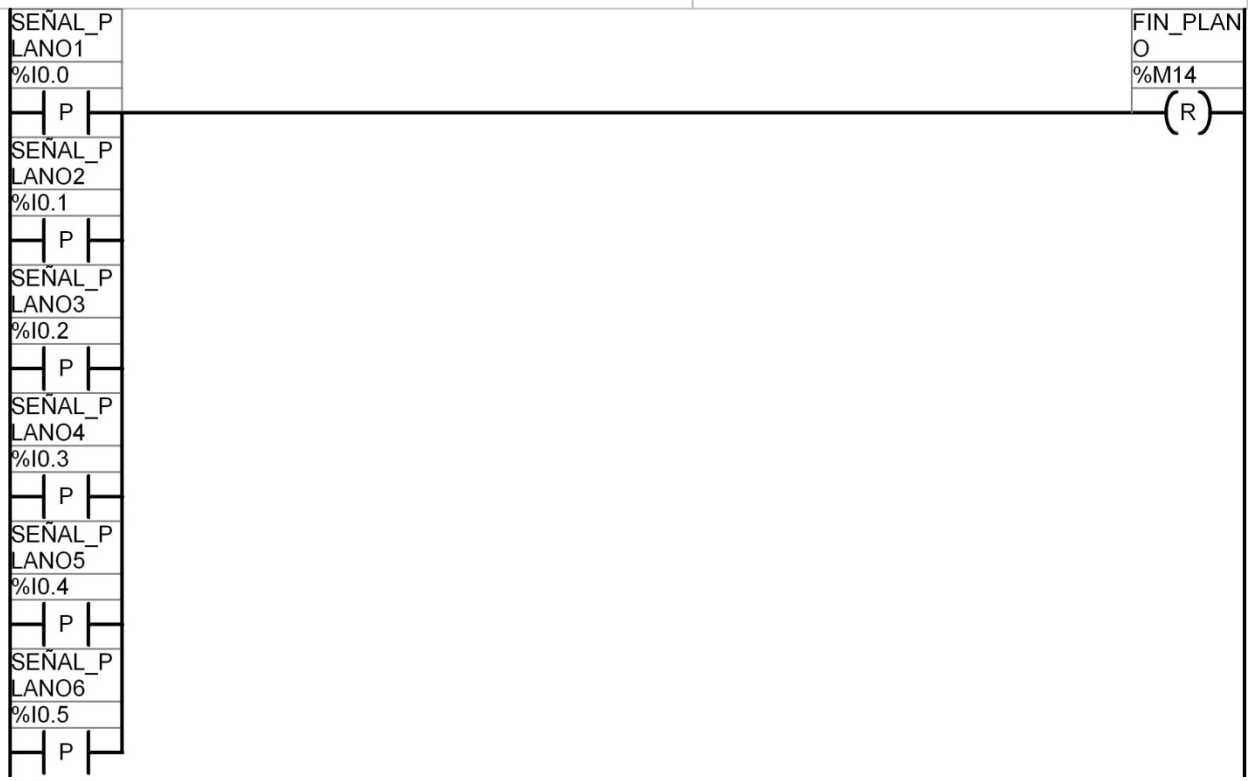



Rung 42
FIN_PROCESO
%M25
INDICADOR_SEQ := 0
%MW13 := 0
Rung 43
SEÑAL_PLANO1
%I0.0
PLANO1_PLC
%M15
NUM_PLANO := 1
%MW15 := 1
Rung 44
SEÑAL_PLANO2
%I0.1
PLANO2_PLC
%M16
NUM_PLANO := 2
%MW15 := 2
Rung 45
SEÑAL_PLANO3
%I0.2
PLANO3_PLC
%M17
NUM_PLANO := 3
%MW15 := 3
Rung 46
SEÑAL_PLANO4
%I0.3
PLANO4_PLC
%M18
NUM_PLANO := 4
%MW15 := 4
Rung 47
SEÑAL_PLANO5
%I0.4
PLANO5_PLC
%M19
NUM_PLANO := 5
%MW15 := 5
Rung 48
SEÑAL_PLANO6
%I0.5
PLANO6_PLC
%M20
NUM_PLANO := 6
%MW15 := 6

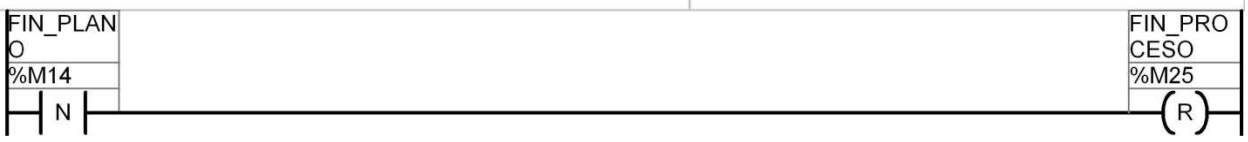
Rung 49

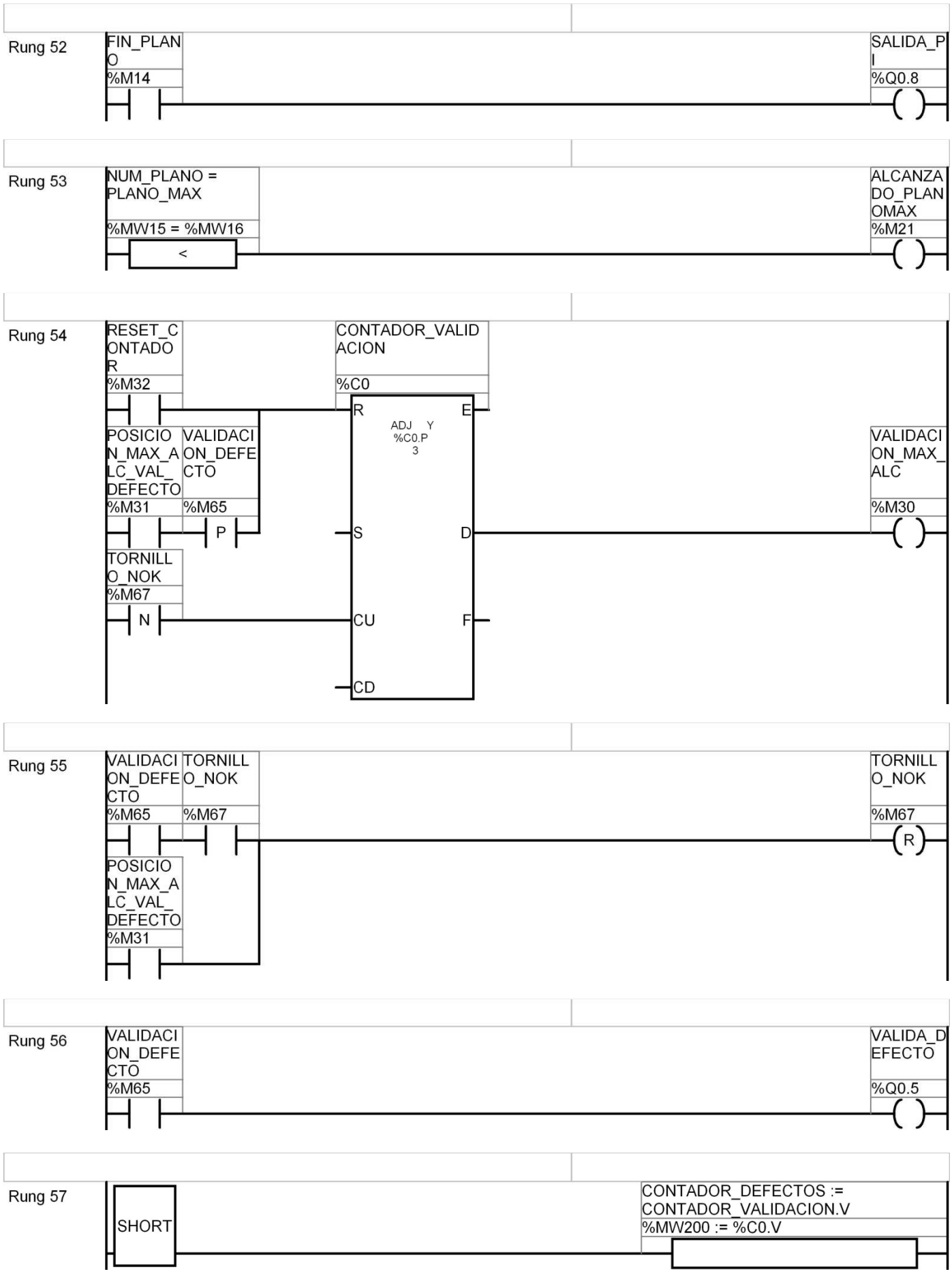


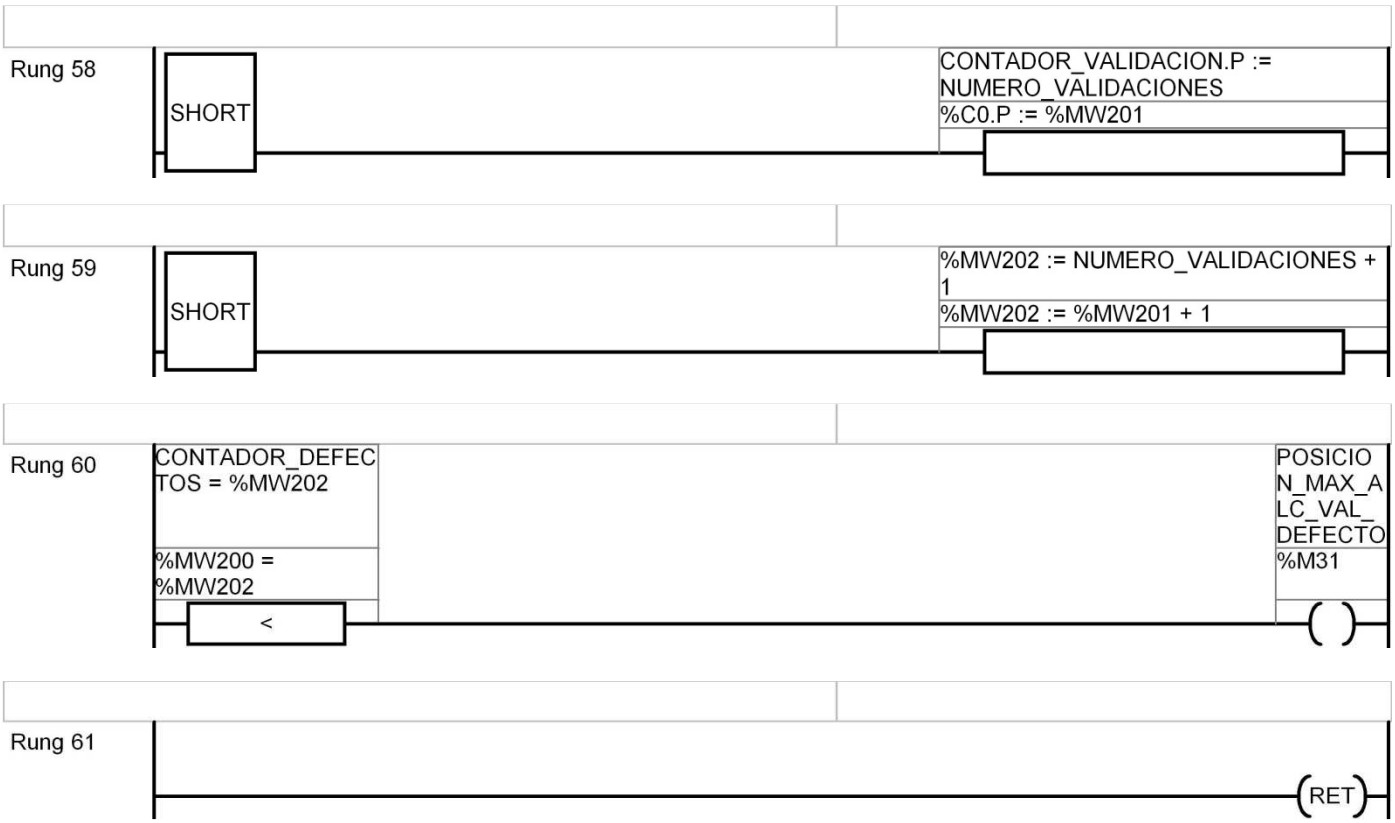
Rung 50



Rung 51




Rung 52
FIN_PLANO
%M14
SALIDA_PI
%Q0.8
Rung 53
NUM_PLANO = PLANO_MAX
%MW15 = %MW16
<
ALCANZADO_PLANOMAX
%M21
Rung 54
RESET_CONTADOR
%M32
CONTADOR_VALIDACION
%C0
R
E
ADJ Y
%C0.P
3
POSICION_MAX_ALC_VAL_DEFECTO
%M31
VALIDACION_DEFECTO
%M65
P
S
D
VALIDACION_MAX_ALC
%M30
TORNILLO_NOK
%M67
N
CU
F
CD
Rung 55
VALIDACION_DEFECTO
%M65
TORNILLO_NOK
%M67
TORNILLO_NOK
%M67
R
POSICION_MAX_ALC_VAL_DEFECTO
%M31
Rung 56
VALIDACION_DEFECTO
%M65
VALIDA_DEFECTO
%Q0.5
Rung 57
SHORT
CONTADOR_DEFECTOS := CONTADOR_VALIDACION.V
%MW200 := %C0.V

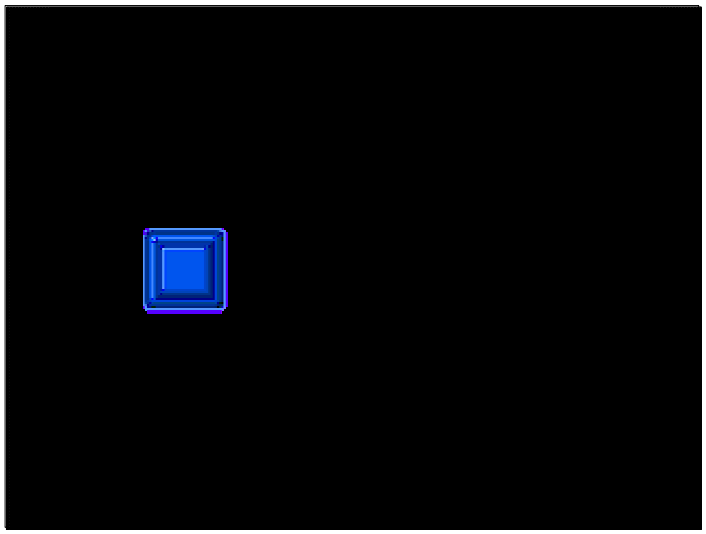

Rung 58
SHORT
CONTADOR_VALIDACION.P := NUMERO_VALIDACIONES
%C0.P := %MW201
Rung 59
SHORT
%MW202 := NUMERO_VALIDACIONES + 1
%MW202 := %MW201 + 1
Rung 60
CONTADOR_DEFECTOS = %MW202
%MW200 = %MW202
<
POSICION_MAX_ALC_VAL_DEFECTO
%M31
Rung 61
RET

Notas:

ANEXO II: Programación y descripción del Terminal HMI en Vijeo Designer

DESCRIPCIÓN DE LOS PANELES Y PROGRAMACIÓN

Panel 1: Principal



Funciones del panel: PRINCIPAL

1 Periódica - Repetir cada 1 seg.
Script

```
int manual;
manual=MAN_AUT.getIntValue();
if (manual==1)
{
   _CurPanelID.write(10);

}
```

2 Evento - Al abrir
Script

```
int uno=1,cero=0;

INDICADOR_SEQ.write(cero);
CICLO_ATORNILLADOR.write(cero);

if (PLANOS_MAX.getIntValue()==0)
{
   PLANOS_MAX.write(3);
}
```

3 Evento - Al cerrar
Script

```
int
tornillo1,tornillo2,tornillo3,tornillo4,tornillo5,tornillo6,uno=1;
```

```
String cadena01,cadena02,cadena03;
cadena01=NOM_PLANO1.getStringValue();
cadena02=NOM_PLANO2.getStringValue();
cadena03=NOM_PLANO3.getStringValue();

tornillo1=MAX_TORNILLOS_1.getIntValue();
tornillo2=MAX_TORNILLOS_2.getIntValue();
tornillo3=MAX_TORNILLOS_3.getIntValue();
tornillo4=MAX_TORNILLOS_4.getIntValue();
tornillo5=MAX_TORNILLOS_5.getIntValue();
tornillo6=MAX_TORNILLOS_6.getIntValue();
if (tornillo1==0)
{
MAX_TORNILLOS_1.write(uno);
}
if (tornillo2==0)
{
MAX_TORNILLOS_2.write(uno);
}

if (tornillo3==0)
{
MAX_TORNILLOS_3.write(uno);
}

if (tornillo4==0)
{
MAX_TORNILLOS_4.write(uno);
}

if (tornillo5==0)
{
MAX_TORNILLOS_5.write(uno);
}

if (tornillo6==0)
{
MAX_TORNILLOS_6.write(uno);
}

if (cadena01=="")
{
   cadena01="Casting de Entrada";
   NOM_PLANO1.write(cadena01);

}

if (cadena02=="")
```

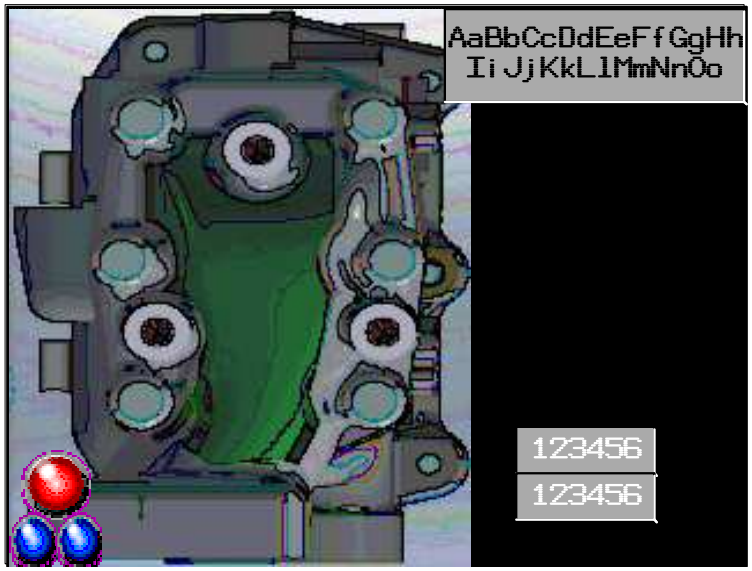
```
{
   cadena02="Casting de salida";
   NOM_PLANO2.write(cadena02);

}

if (cadena03=="")
{
   cadena03="Protector Calor";
   NOM_PLANO3.write(cadena03);

}
```

Panel 2: DISEÑO_PIEZA_PLANO1

Funciones del panel: DISEÑO_PIEZA_PLANO1

1 Evento - Al abrir
Set palabra [NUM_PLANO] = [1]
Set palabra [MAX_TORNILLOS_1] = [6]

Panel 3: PIEZA_PLANO1



Funciones del panel: PIEZA_PLANO1

1 Periódica - Repetir cada 0,5 seg.
Script

```
int end_plano, fin, tornillo_no_ok, man, maximo_validacion, contador,valida;
tornillo_no_ok=tornillo_defectuoso.getIntValue();
man= MAN_AUT.getIntValue();
end_plano=FIN_PLANO.getIntValue();
fin=0;
maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();

if (end_plano==1)
{
    Ventana1.open(100,100);

    Ventana1.changePanel(10003);

    Ventana1.show(true);

    pieza_ok.write(fin);

}


if (man==0)
{
```

```
    _CurPanelID.write(1);
}

if ((valida!=1)&&(tornillo_no_ok==1))
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10011);

    Ventana1.show(true);
}

if ((valida==1)&&(tornillo_no_ok==1))
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10015);

    Ventana1.show(true);
}

maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();
```

2 Evento - Al abrir
Script

```
int varX1=0, varX2=0, varX3=0,varY1=0, varY2=0, varY3=0,
varX4=0, varY4=0, varX5=0, varY5=0, varX6=0, varY6=0,seq,NUM,maxtor;
NUM=NUM_PLANO.getIntValue();
maxtor=MAX_TORNILLOS_1.getIntValue();
MAX_TORNILLOS.write(maxtor);

varX1 = INTX1.getIntValue();
PLC_X1.write(varX1);

varY1 = INTY1.getIntValue();
PLC_Y1.write(varY1);

varX2 = INTX2.getIntValue();
PLC_X2.write(varX2);

varY2 = INTY2.getIntValue();
PLC_Y2.write(varY2);

varX3 = INTX3.getIntValue();
PLC_X3.write(varX3);
```

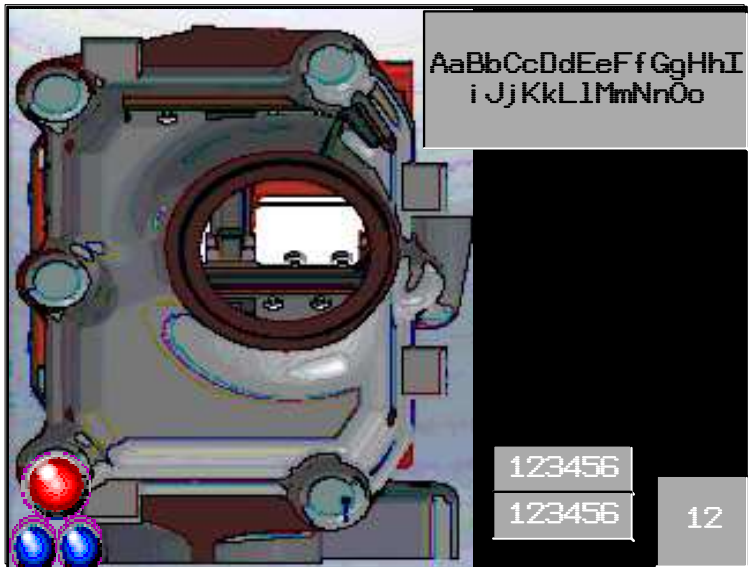
```
varY3 = INTY3.getIntValue();
PLC_Y3.write(varY3);

varX4 = INTX4.getIntValue();
PLC_X4.write(varX4);

varY4 = INTY4.getIntValue();
PLC_Y4.write(varY4);

varX5 = INTX5.getIntValue();
PLC_X5.write(varX5);

varY5 = INTY5.getIntValue();
PLC_Y5.write(varY5);

varX6 = INTX6.getIntValue();
PLC_X6.write(varX6);

varY6 = INTY6.getIntValue();
PLC_Y6.write(varY6);

//ciclo
int cic1=0,cic2=0,cic3=0,cic4=0,cic5=0,cic6=0;

cic1=CICLO1.getIntValue();
CICLO1_PLC.write(cic1);

cic2=CICLO2.getIntValue();
CICLO2_PLC.write(cic2);

cic3=CICLO3.getIntValue();
CICLO3_PLC.write(cic3);

cic4=CICLO4.getIntValue();
CICLO4_PLC.write(cic4);

cic5=CICLO5.getIntValue();
CICLO5_PLC.write(cic5);

cic6=CICLO6.getIntValue();
CICLO6_PLC.write(cic6);

seq=1;
INDICADOR_SEQ.write(seq);
```

Panel 4: DISEÑO_PIEZA_PLANO2

Funciones del panel: DISEÑO_PIEZA_PLANO2

1 Evento - Al abrir
Set palabra [NUM_PLANO] = [2]
Set palabra [MAX_TORNILLOS_2] = [5]

Panel 5: PIEZA_PLANO2



Funciones del panel: PIEZA_PLANO2

1 Periódica - Repetir cada 0,5 seg.

Script

```
int end_plano, fin, tornillo_no_ok, man, maximo_validacion, contador,valida;
tornillo_no_ok=tornillo_defectuoso.getIntValue();
man= MAN_AUT.getIntValue();
end_plano=FIN_PLANO.getIntValue();
fin=0;
maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();

if (end_plano==1)
{
    Ventana1.open(100,100);

    Ventana1.changePanel(10003);

    Ventana1.show(true);

    pieza_ok.write(fin);

}
if (man==0)
{
   _CurPanelID.write(1);
}

if ((valida!=1)&&(tornillo_no_ok==1))
```

```
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10011);

    Ventana1.show(true);
}

if ((valida==1)&&(tornillo_no_ok==1))
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10015);

    Ventana1.show(true);
}

maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();
```

2 Evento - Al abrir

Script

```
int varX1=0, varY1=0, varX2=0, varY2=0, varX3=0, varY3=0,
varX4=0, varY4=0, varX5=0, varY5=0, varX6=0, varY6=0,seq,NUM,lim_tor2;
NUM=NUM_PLANO.getIntValue();

MAX_TORNILLOS.write(5);
varX1 = INTX1_2.getIntValue();
PLC_X1.write(varX1);

varY1 = INTY1_2.getIntValue();
PLC_Y1.write(varY1);

varX2 = INTX2_2.getIntValue();
PLC_X2.write(varX2);

varY2 = INTY2_2.getIntValue();
PLC_Y2.write(varY2);

varX3 = INTX3_2.getIntValue();
PLC_X3.write(varX3);

varY3 = INTY3_2.getIntValue();
PLC_Y3.write(varY3);

varX4 = INTX4_2.getIntValue();
PLC_X4.write(varX4);
```

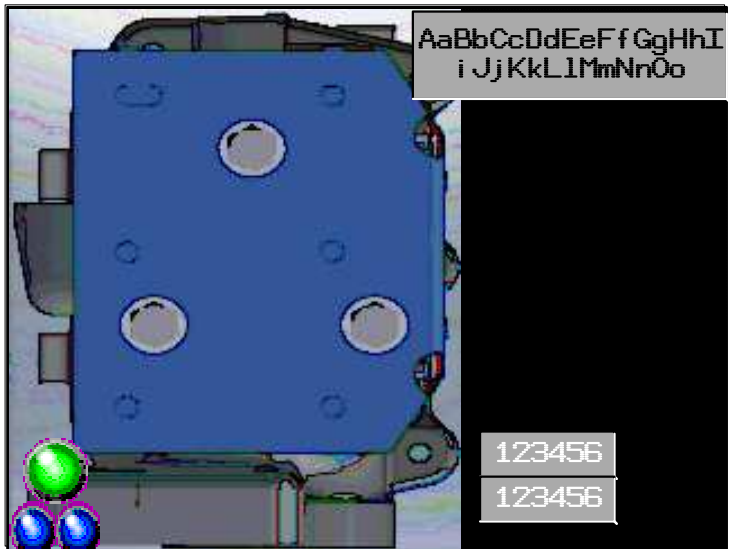
```
varY4 = INTY4_2.getIntValue();
PLC_Y4.write(varY4);

varX5 = INTX5_2.getIntValue();
PLC_X5.write(varX5);

varY5 = INTY5_2.getIntValue();
PLC_Y5.write(varY5);

varX6 = INTX6_2.getIntValue();
PLC_X6.write(varX6);

varY6 = INTY6_2.getIntValue();
PLC_Y6.write(varY6);

//ciclo
int cic1=0,cic2=0,cic3=0,cic4=0,cic5=0,cic6=0;

cic1=CICLO1_2.getIntValue();
CICLO1_PLC.write(cic1);

cic2=CICLO2_2.getIntValue();
CICLO2_PLC.write(cic2);

cic3=CICLO3_2.getIntValue();
CICLO3_PLC.write(cic3);

cic4=CICLO4_2.getIntValue();
CICLO4_PLC.write(cic4);

cic5=CICLO5_2.getIntValue();
CICLO5_PLC.write(cic5);

cic6=CICLO6_2.getIntValue();
CICLO6_PLC.write(cic6);

seq=1;
INDICADOR_SEQ.write(seq);
```

Panel 6: DISEÑO_PIEZA_PLANO3

Funciones del panel: DISEÑO_PIEZA_PLANO3

1 Evento - Al abrir
Set palabra [NUM_PLANO] = [3]
Set palabra [MAX_TORNILLOS_3] = [3]

Panel 7: PIEZA_PLANO3



Funciones del panel: PIEZA_PLANO3

1 Periódica - Repetir cada 0,5 seg.
Script

```
int end_plano, fin, tornillo_no_ok, man, maximo_validacion,
contador,valida;
tornillo_no_ok=tornillo_defectuoso.getIntValue();
man= MAN_AUT.getIntValue();
end_plano=FIN_PLANO.getIntValue();
fin=0;
maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();

if (end_plano==1)
{
    Ventana1.open(100,100);

    Ventana1.changePanel(10003);

    Ventana1.show(true);

    pieza_ok.write(fin);
}
if (man==0)
{
   _CurPanelID.write(1);
}

if ((valida!=1)&&(tornillo_no_ok==1))
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10011);

    Ventana1.show(true);
}

if ((valida==1)&&(tornillo_no_ok==1))
{
   Ventana1.open(100,100);

    Ventana1.changePanel(10015);

    Ventana1.show(true);
}

maximo_validacion=VALOR_DEFECTO.getIntValue();
contador=CONTADOR_DEFECTO.getIntValue();
valida=validacion_alcanzada.getIntValue();
```

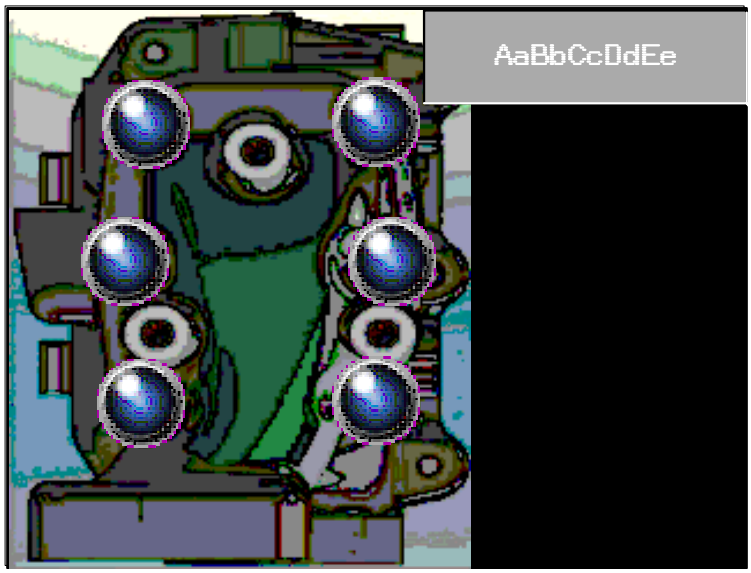
2 Evento - Al abrir

Script

```
int varX1=0, varY1=0, varX2=0, varY2=0, varX3=0, varY3=0, varX4=0, varY4=0, varX5=0, varY5=0, varX6=0, varY6=0,seq,NUM,lim_tor2;
NUM=NUM_PLANO.getIntValue();

MAX_TORNILLOS.write(3);
varX1 = INTX1_3.getIntValue();
PLC_X1.write(varX1);

varY1 = INTY1_3.getIntValue();
PLC_Y1.write(varY1);

varX2 = INTX2_3.getIntValue();
PLC_X2.write(varX2);

varY2 = INTY2_3.getIntValue();
PLC_Y2.write(varY2);

varX3 = INTX3_3.getIntValue();
PLC_X3.write(varX3);

varY3 = INTY3_3.getIntValue();
PLC_Y3.write(varY3);

varX4 = INTX4_3.getIntValue();
PLC_X4.write(varX4);

varY4 = INTY4_3.getIntValue();
PLC_Y4.write(varY4);

varX5 = INTX5_3.getIntValue();
PLC_X5.write(varX5);

varY5 = INTY5_3.getIntValue();
PLC_Y5.write(varY5);

varX6 = INTX6_3.getIntValue();
PLC_X6.write(varX6);

varY6 = INTY6_3.getIntValue();
PLC_Y6.write(varY6);

//ciclo
int cic1=0,cic2=0,cic3=0,cic4=0,cic5=0,cic6=0;

cic1=CICLO1_3.getIntValue();
CICLO1_PLC.write(cic1);
```

```
cic2=CICLO2_3.getIntValue();
CICLO2_PLC.write(cic2);

cic3=CICLO3_3.getIntValue();
CICLO3_PLC.write(cic3);

cic4=CICLO4_3.getIntValue();
CICLO4_PLC.write(cic4);

cic5=CICLO5_3.getIntValue();
CICLO5_PLC.write(cic5);

cic6=CICLO6_3.getIntValue();
CICLO6_PLC.write(cic6);

seq=1;
INDICADOR_SEQ.write(seq);
```

Panel 8: POSICIONES_PLANO1



Funciones del panel: POSICIONES_PLANO1

1 Periódica - Repetir cada 1 seg.
Enclavamiento: secuencia_act==7&&POSP1X6!=0&&POSP1Y6!=0
Cambiar panel[2]

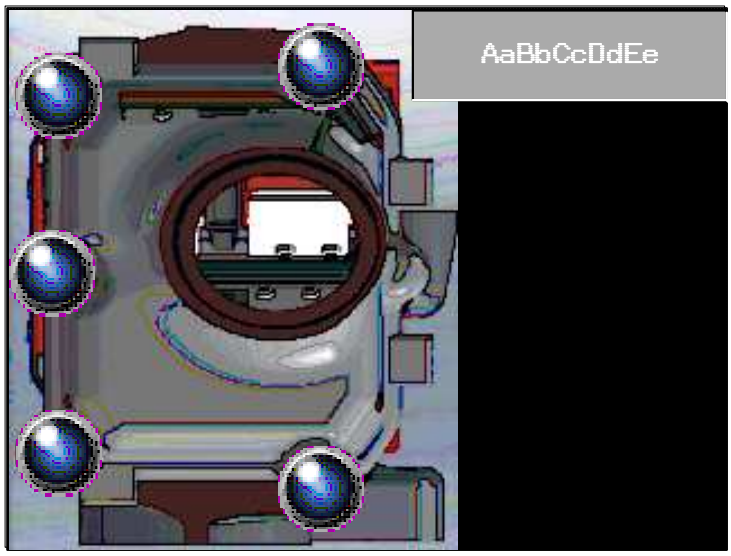
2 Condicional - Si es falso, MAN_AUT
Enclavamiento: MAN_AUT==0
Cambiar panel[1]

3 Evento - Al abrir
Set bit [visibleP1B1]
Set bit [visibleP1B2]
Set bit [visibleP1B3]
Set bit [visibleP1B4]
Set bit [visibleP1B5]
Set bit [visibleP1B6]
Set palabra [secuencia_act] = [1]
Script

```
POSP1X1.write(0);
POSP1X2.write(0);
POSP1X3.write(0);
POSP1X4.write(0);
POSP1X5.write(0);
POSP1X6.write(0);

POSP1Y1.write(0);
POSP1Y2.write(0);
```

POSP1Y3.write(0);
POSP1Y4.write(0);
POSP1Y5.write(0);
POSP1Y6.write(0);

Panel 9: POSICIONES_PLANO2



Funciones del panel: POSICIONES_PLANO2

1 **Periódica - Repetir cada 1 seg.**
Enclavamiento: secuencia_act02==6&&POSP2X5!=0&&POSP2Y5!=0
Cambiar panel[4]

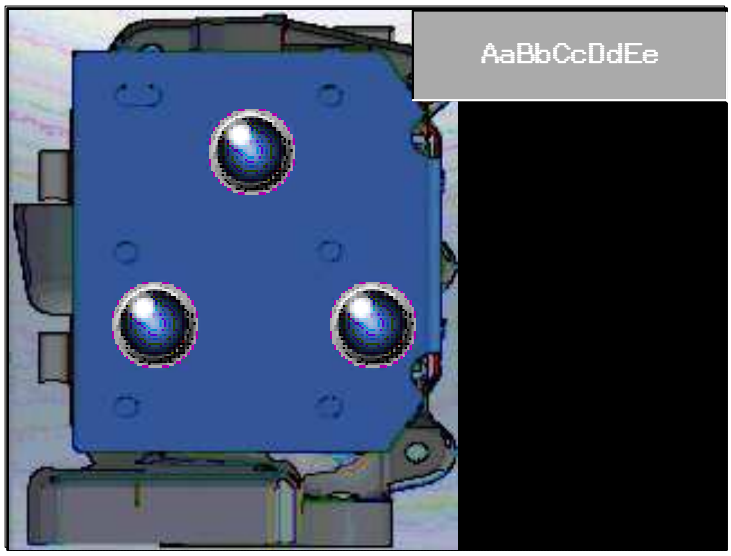
2 **Condicional - Si es falso, MAN_AUT**
Enclavamiento: MAN_AUT==0
Cambiar panel[1]

3 **Evento - Al abrir**
Set bit [visibleP2B1]
Set bit [visibleP2B2]
Set bit [visibleP2B3]
Set bit [visibleP2B4]
Set bit [visibleP2B5]
Set bit [visibleP2B6]
Set palabra [secuencia_act02] = [1]
Script

POSP2X1.write(0);
POSP2X2.write(0);
POSP2X3.write(0);
POSP2X4.write(0);

```
POSP2X5.write(0);
POSP2X6.write(0);

POSP2Y1.write(0);
POSP2Y2.write(0);
POSP2Y3.write(0);
POSP2Y4.write(0);
POSP2Y5.write(0);
POSP2Y6.write(0);
```

Panel 9: POSICIONES_PLANO3



Funciones del panel: POSICIONES_PLANO3

1 **Periódica - Repetir cada 1 seg.**
Enclavamiento: secuencia_act03==4&&POSP3X3!=0&&POSP3Y3!=0
Cambiar panel[8]

2 **Condicional - Si es falso, MAN_AUT**
Enclavamiento: MAN_AUT==0
Cambiar panel[1]

3 **Evento - Al abrir**
Set bit [visibleP3B1]
Set bit [visibleP3B2]
Set bit [visibleP3B3]
Set bit [visibleP3B4]
Set bit [visibleP3B5]
Set bit [visibleP3B6]
Set palabra [secuencia_act03] = [1]
Script

```
POSP3X1.write(0);
POSP3X2.write(0);
POSP3X3.write(0);
POSP3X4.write(0);
POSP3X5.write(0);
POSP3X6.write(0);

POSP3Y1.write(0);
POSP3Y2.write(0);
POSP3Y3.write(0);
POSP3Y4.write(0);
POSP3Y5.write(0);
POSP3Y6.write(0);
```

Panel 10: ESPERA INFORMACION PLANO

Funciones del panel: **ESPERA INFORMACION PLANO**

1 Periódica - Repetir cada 1 seg.

Script

```
int plano1, plano2, plano3, plano4, plano5, plano6, planomax,numplan,cero,manual;
plano1=PLANO1_PLC.getIntValue();
plano2=PLANO2_PLC.getIntValue();
plano3=PLANO3_PLC.getIntValue();
plano4=PLANO4_PLC.getIntValue();
plano5=PLANO5_PLC.getIntValue();
plano6=PLANO6_PLC.getIntValue();
planomax=PLANOS_MAX.getIntValue();
manual=MAN_AUT.getIntValue();
```

```
if ((plano1==1))

{
   numplan=1;
   _CurPanelID.write(3);
   NUM_PLANO.write(numplan);
}
if ((plano2==1)&(planomax>1))
{
   numplan=2;
   _CurPanelID.write(11);
   NUM_PLANO.write(numplan);
int seq;
seq=1;
INDICADOR_SEQ.write(seq);

}
if ((plano3==1)&(planomax>2))
{
   numplan=3;
   _CurPanelID.write(9);
   NUM_PLANO.write(numplan);

int seq;
seq=1;
INDICADOR_SEQ.write(seq);

}

if ((plano4==1)&(planomax>3))
{
   numplan=4;
   _CurPanelID.write(13);
   NUM_PLANO.write(numplan);

int seq;
seq=1;
INDICADOR_SEQ.write(seq);

}

if ((plano5==1)&(planomax>4))
{
   numplan=5;
   _CurPanelID.write(15);
   NUM_PLANO.write(numplan);

int seq;
```

```
seq=1;
INDICADOR_SEQ.write(seq);
                }
if ((plano6==1)&(planomax>5))
{
   numplan=6;
   _CurPanelID.write(17);
   NUM_PLANO.write(numplan);

int seq;
seq=1;
INDICADOR_SEQ.write(seq);
 }

if (manual==0)
{
   _CurPanelID.write(1);
}
```

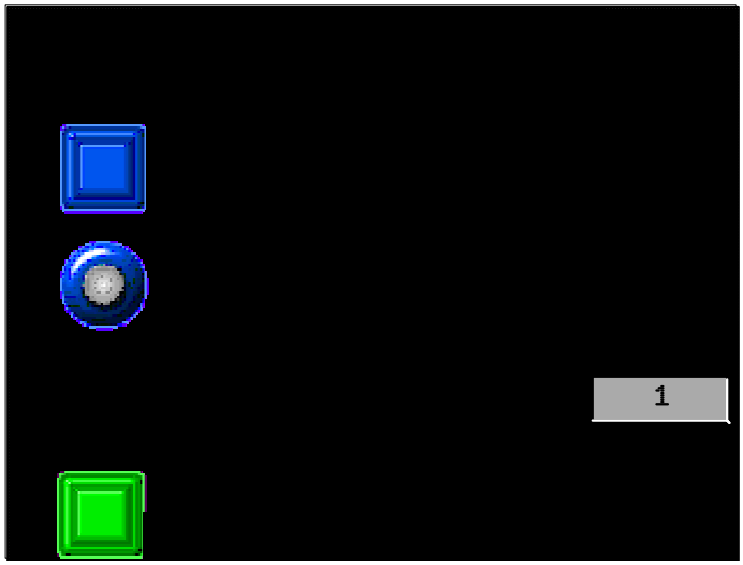
2 Evento - Al abrir

Script

```
int cero, manual;
cero=0;
manual=MAN_AUT.getIntValue();
CICLO1_PLC.write(cero);
CICLO2_PLC.write(cero);
CICLO3_PLC.write(cero);
CICLO4_PLC.write(cero);
CICLO5_PLC.write(cero);
CICLO6_PLC.write(cero);
PLC_X1.write(cero);
PLC_X2.write(cero);
PLC_X3.write(cero);
PLC_X4.write(cero);
PLC_X5.write(cero);
PLC_X6.write(cero);
PLC_Y1.write(cero);
PLC_Y2.write(cero);
PLC_Y3.write(cero);
PLC_Y4.write(cero);
PLC_Y5.write(cero);
PLC_Y6.write(cero);

if (manual==0)
{
   _CurPanelID.write(1);
}
```

3 Evento - Al cerrar
Reset bit [pieza_mala]
Reset bit [VALIDACION_DEFECTO]

Panel 11: CONFIGURACIÓN



Funciones del panel: CONFIGURACION

Panel 12: NOMBRE PLANOS

Funciones del panel: NOMBRE PLANOS

VENTANAS EMERGENTES

OFFSET

Debido a que la posición real nunca suele coincidir con la programada , se configura una pantalla con histéresis para crear una ventana de ajuste.

Nº TORNILLOS PLANO 1

Se especifica el número de tornillos que contiene el PLANO 1.

1 Periódica - Repetir cada 0,5 seg.
Script
int tornillos;
tornillos=MAX_TORNILLOS_1_INTERNO.getIntValue();
MAX_TORNILLOS_1.write(tornillos);

Nº TORNILLOS PLANO 2

Se especifica el número de tornillos que contiene el PLANO 2.

1 Periódica - Repetir cada 0,5 seg.
Script
int tornillos;
tornillos=MAX_TORNILLOS_2_INTERNO.getIntValue();
MAX_TORNILLOS_2.write(tornillos);

Nº TORNILLOS PLANO 3

Se especifica el número de tornillos que contiene el PLANO 3.

1 Periódica - Repetir cada 0,5 seg.
Script
int tornillos;
tornillos=MAX_TORNILLOS_3_INTERNO.getIntValue();
MAX_TORNILLOS_3.write(tornillos);

ELECCION DE CICLO

En esta pantalla se escoge el ciclo de funcionamiento programado en el sistema de atonillado automático.

PROCESO CORRECTO

Se indica que el proceso de atornillado ha sido correcto. Esta información la da el sistema de atornillado automático.

PROCESO INCORRECTO

En cualquiera de las fases de atornillado de un plano puede fallar el proceso, el cual obliga a comenzar de nuevo el proceso por el primer tornillo.

SEGURIDAD Y MANTENIMIENTO

Se le puede indicar al sistema una clave de acceso para la modificación de los parámetros.

Notas: